UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

A 3D PIPELINE FOR 2D PIXEL ART ANIMATION

BEATRIZ GOMES DA COSTA

Thesis supervisor: RUBÉN TOUS LIESA (Department of Computer Architecture)

Degree: Bachelor Degree in Informatics Engineering (Computing)

Thesis report

Facultat d'Informàtica de Barcelona (FIB)

Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

27/06/2023

Abstract

This document presents a comprehensive report on a project aimed at developing an automated process for creating 2D animations from 3D models using Blender. The project's main goal is to improve upon existing techniques and reduce the need for artists to do clerical tasks in the animation production process. The project involves the design and development of a plugin for Blender, coded in Python, which was developed to be efficient and reduce time-intensive tasks that usually characterise some stages in the animation process. The plugin supports three specific styles of animation: pixel art, cel shading, and cel shading with outlines, and can be expanded to support a wider range of styles. The plugin is also open-source, allowing for greater collaboration and potential contributions from the community. Despite the challenges faced, the project was successful in achieving its goals, and the results show that the plugin could achieve results similar to those acquired with similar tools and traditional animation. The future work includes keeping the plugin up-to-date with the latest versions of Blender, publishing it on GitHub and Blender plugin markets, as well as adding new art styles.

Abstract

Aquest document presenta un informe exhaustiu sobre un projecte destinat a desenvolupar un procés automatitzat per a la creació d'animacions 2D a partir de models 3D utilitzant Blender. L'objectiu principal del projecte és millorar les tècniques existents i reduir la necessitat que els artistes realitzin tasques repetitives en el procés de producció d'animació. El projecte implica el disseny i desenvolupament d'un complement per a Blender, programat en Python, que es va desenvolupar per ser eficient i reduir les tasques intensives en temps que solen caracteritzar algunes etapes en el procés d'animació. El complement suporta tres estils específics d'animació: l'art de píxel, "cel shader", i "cel shader" amb contorns, i es pot expandir per suportar una àmplia gamma d'estils. El complement també és de codi obert, permetent una major col·laboració i potencials contribucions per part de la comunitat. Malgrat els problemes trobats, el projecte ha estat exitós en aconseguir els seus objectius, i els resultats mostren que el complement pot aconseguir resultats similars als adquirits amb eines similars i animació tradicional. El treball futur inclou mantenir el complement actualitzat amb les últimes versions de Blender, publicar-lo a GitHub i mercats de complements de Blender, així com afegir nous estils d'art.

Contents

List of Figures

List of Tables

1 Context and Scope

1.1 Introduction and Contextualization

"Animation is a method by which still figures are manipulated to appear as moving images."[1] In other words, the animation is defined as the process of providing movement to static models, objects and sprites. It is an essential step in different disciplines, such as video game development, animation, and modelling. Although generally, the term has been used to reference the more classical two-dimensional animation, it currently includes 3D animation as well. Given the specific needs of 2D and 3D animations, the usual pipelines used to achieve results in both of them differ significantly, with the two-dimensional pipelines usually being the most time-consuming ones.

Two-dimensional animation has proven to be a very complex process since its very beginning. In order to achieve the desired movement, single frames had to be drawn one at a time, by hand. The arrival of the first drawing software was achieved to speed up the process. However, single frames still had to be drawn and the slightest change would still mean a considerable load of work for the animator. Additionally, 2D animation requires good drawing skills. Leaving the task out of reach to a considerably large amount of people. The current challenges found in workflows for 3D animation are more related to the steeper learning curve for most modelling software. However, modelling 3D objects can be just as time-consuming as drawing in 2D.

Recent advances in animation workflow tools are focusing on the enlargement of this clear bottleneck in the processes that require the animation step. Methods such as pose interpolation [2], motion capture [3] and Artificial Intelligence [4] have shown great improvements in speeding up this task. Most of these are already very popularized, and there are some remarkable examples that use these modern techniques.

Combining aspects of both 2D and 3D pipelines for animation is becoming more and more common [5]. This is not only done to help improve the overall process by avoiding the most time-consuming steps of each workflow, although this alone can be reason enough. It can also be used to achieve a more unique result. This combined pipeline is used for producing both two-dimensional and three-dimensional results.

In this thesis, we are going to implement a tool for automating the process of turning an animated 3D model into 2D animation with different art styles, in order to contribute to the facilitation of the overall animating process. Since there is a large variety of modelling software available, we will focus on implementing this tool for a specific open-source modelling tool named Blender[6]. By choosing this specific tool, we are ensuring that the final result reaches an already notably large community[7], in a completely free way.

1.1.1 Context

This Bachelor's Thesis of the Computer Engineering Degree with a specialization in Computing, done in the Informatics Faculty of the Polytechnic University of Catalonia. The thesis is directed by Rubèn Tous Liesa.

1.1.2 Concepts

This thesis focuses on the automatization of the animating process. The concepts outlined below provide the preliminary knowledge required to properly understand this dissertation.

Animation workflows

The concept of Animation workflows refers to the step-by-step processes that animators follow in order to create an animated sequence. Typically, these workflows involve a series of stages. Their aim is to take a project from concept to completion. As has already been mentioned, the exact workflow can vary depending on the project, the style of animation, and the software and tools being used. However, below a general overview of the typical stages in a 2D animation workflow is shown.

- **Pre-production:** Involves all of the planning and preparation that goes into a project before any animation takes place. This can include storyboarding, character design, script writing, and creating animatics.
- **Production:** Where the actual animation takes place. This can involve creating keyframes, inbetweening, adding colour and shading, and adding special effects and even some sound effects.
- **Post-production:** This stage involves editing and polishing the final animated sequence. This can include editing the timing, adding final sound effects and music, and compositing the final sequence.

Once these steps have been completed, the final result generally is either a set of frames that compose the animation or a format video file. The final animation can be the final project itself, or it can also belong to a larger project, such as a video game for which said animations will be used.

2D and 3D animation approaches

Multiple processes can be followed during the production stage of the animation workflow. In this thesis, some of the most common approaches will be mentioned and discussed.

Straight ahead
This is the most basic approach and consists in simply drawing all the animation frames, one by one.

Pose to pose
This methodology involves drawing the extreme poses first and then finishing by filling the sequence with the in-between frames. This gives to the animator an overall idea of how the final animation will look earlier on in the process and provides better control regarding the total number of frames as well as the timing. Additionally, post-process alterations can be applied easily. Pose by pose is a very common method for animating sprites since they usually transition between a few important poses.

Interpolation
Similar to the previous method, interpolation starts by drawing or modelling the important poses. However, once this has been done, some interpolation tool is used to create the transitions between them. This can be one of the most time-efficient ways of coming up with animations, but it also provides less control over the transitioning frames.

Motion cap
Motion capture is a technology used to capture and record the movements of people or objects in real-time. It is used in a variety of fields, such as film, video games, sports, and medicine. The process consists in placing markers on the subject's body or an object and tracking their movement with cameras and sensors. This allows the creation of a 3D representation of the movement. The collected data can be used to animate digital characters, analyze athletic performance, or even assist in medical rehabilitation. This technology is also known by the name of mo-Cap.
Recent advances in motion capture technology have focused on improving the accuracy and speed of the capture process. This has been achieved through the use of advanced cameras and sensors that can capture a larger

amount of data points and track movement in greater detail. Additionally, machine learning and artificial intelligence techniques have been applied to motion capture data to improve the accuracy of the resulting animations.

Artificial Intelligence
One of the most recent approaches in animation involves the use of AI for generating animations, especially for video game objects. Instead of creating a predefined sequence of frames or poses, this method's aim is to understand how the object should behave given different events. We understand as events any phenomenon that has to provoke some sort of movement in the object. That includes walking, jumping, collisions with other objects, and behaviour on different fluids (air, water, etc). Undoubtedly, this is one of the most complex approaches. However, recent projects featuring this method have shown how much potential it has since extremely natural and unique results can be achieved.

3D Modeling

The process of creating a three-dimensional representation of an object or character is named 3D modelling. This technique uses specialized software, such as Blender, Maya, or Adobe Illustrator. The process of 3D modelling usually involves the creation of a digital model from scratch. Nonetheless, with the right technology, it can also be achieved by scanning a physical object to create a digital version.

Rigging

Rigging is the process of adding a digital skeleton to a 3D model. Skeletons are made out of a set of bones; special objects that, once they are bound to 3D models, allow them to be animated. Bones come with a system of joints and controls through which the bound model parts can be manipulated and hence, moved, as desired.

Given the potential uniqueness of each 3D model, the process of rigging an object can take up a considerable amount of time. Still there exist different ways of speeding up the rigging process, especially for commonly shaped 3D models. For instance, most 3D modelling software tools for humanoid shapes provide default skeletons that can be easily adapted to fit the model. Another useful option is to use auto-rigging tools, which, automatically bind an object to a skeleton. Mixamo, Adobe's free auto-rigging tool is one of the most commonly used. It also provides a wide variety of premade animations that can be downloaded and used for different projects.

Recent advances in motion capture technology have focused on improving the accuracy and speed of the capture process. This has been achieved through the use of advanced cameras and sensors that can capture a larger amount of data points and track movement in greater detail. Additionally, machine learning and artificial intelligence techniques have been applied to motion capture data to improve the accuracy of the resulting animations.

Compositing and Rendering

Compositing is the process of combining multiple visual elements, such as images or video clips, in order to create a final image or sequence. It is a useful technique for different fields, such as television, advertising, and digital art. Compositing is the stage where the creator aims to make a model that's seamless and cohesive. This is achieved by adjusting the colour, lighting, and other visual elements of each component and blending them together.

Rendering, on the other hand, is the process of generating images or animations from a 3D model or scene. It can be used in a variety of fields; film, video games, architecture, and product design, are among them. The process involves calculating the object's appearance or environment by simulating how light interacts with the object's surfaces. Other factors, such as shadows, reflections, and textures, are taken into account as well.

1.1.3 Problem definition

The problem addressed in this thesis is the time-consuming and labour-intensive process of creating 2D animations from 3D models with various art styles. While there are different software tools available for 3D modelling and animation, the process of creating 2D animations from 3D models with different art styles can be challenging and requires significant manual effort.

The goal of this thesis is to develop an automated tool that can simplify and accelerate this process, specifically targeting the Blender community, a widely used open-source 3D modelling tool. And by providing this accessible tool for turning 3D models into 2D animations with different art styles, this research thesis' objective is to facilitate the overall animating process and contribute to the advancement of the field.

1.1.4 Stakeholders

The main stakeholders involved in this thesis are animators. Professional animators, as well as enthusiasts and beginners, can benefit from a faster 2D animation process, given that for most styles it can be a very time-consuming task. Especially for those without experience in the field, or to which the final animation is not the final project itself but a part of it, relying on such a tool can be a key factor in finishing the project quicker.

In addition to contributing to the Blender community of animators, this thesis has the potential to benefit a wider audience, including the open-source community and the animation industry as a whole. The open-source community includes individuals and organizations that support and promote open-source software, who may be interested in the development or use of an open-source tool for converting 3D to 2D animation. Moreover, this includes companies and studios involved in producing 2D animations, who could benefit from a tool that simplifies the production process and reduces the time and costs associated with converting 3D models to 2D animation.

Another group of stakeholders for this thesis are educational institutions, including schools, universities, and training centres that offer courses in 2D animation, 3D modelling, and related fields such as video game development. The proposed tool could be a valuable resource for teaching and learning about the process of converting 3D models to 2D animation with various art styles. This could benefit students and educators by providing them with a practical tool for exploring different animation techniques and creating compelling visual content. Additionally, the tool could potentially be used as a resource for research in the field of computer graphics and animation, further benefiting the academic community.

1.2 Justification

1.2.1 Previous studies

Computer animation production is a time-consuming process that requires both artistic and technical skills. Every animation workflow has its pros and cons, and for many years, several approaches have been proposed to automate, at least partially, these different processes. As has been stated, this thesis will focus on the improvement of workflows that combine 3D and 2D techniques in order to achieve 2D animations. However, tools that are 2D or 3D specific will be mentioned as well in this section, since they are also quite related to this study as well as to the overall goal of making the animating process more time-efficient.

Reallusion's Cartoon Animator [8] allows users to create 2D animations with the help of 3D tools. One of its features is the ability to map motion from a basic 3D model onto a 2D object. This feature is known as "3D motion conversion" and it enables users to take advantage of the benefits of 3D motion data while working in a 2D environment.

Figure 1: Reallusion Cartoon Animator Motion mapping

Adobe's Mixamo [9] tool is primarily designed for motion mapping. It is used to transfer motion data from one character to another, typically between two 3D humanoid models. This is useful for obtaining animations that require specific character movements or interactions, as it comes with a very extensive database of actions

that can be used in all the fields that require animating models.

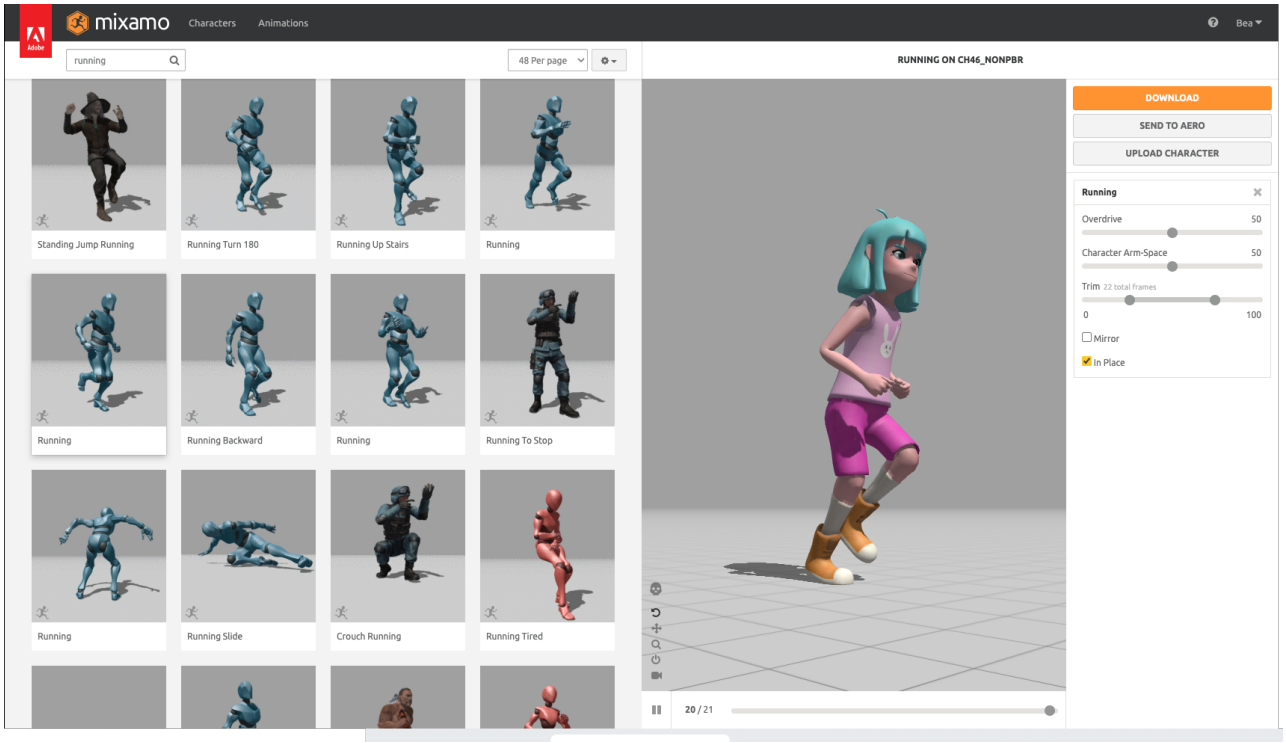

Figure 2: Adobe's Mixamo tool

DeepMotion [10] is a software platform that provides a range of AI-powered tools for animation production. Animate 3D is a motion capture and analysis tool that uses machine learning algorithms to analyze motion data, typically from video, and generate realistic 3D meshes that mimic the captured motion data.

Figure 3: DeepMotion video mapping tool

Cascadeur [11] is a physics-based animation software that uses AI algorithms to enhance the animation production process. One of its main features is its "motion analysis tools", which use AI algorithms to analyze motion data and identify key features, such as joint angles and velocities [12]. This data can be used to create more realistic animations and to help animators work more efficiently by automating some of the more time-consuming aspects of the animation process.

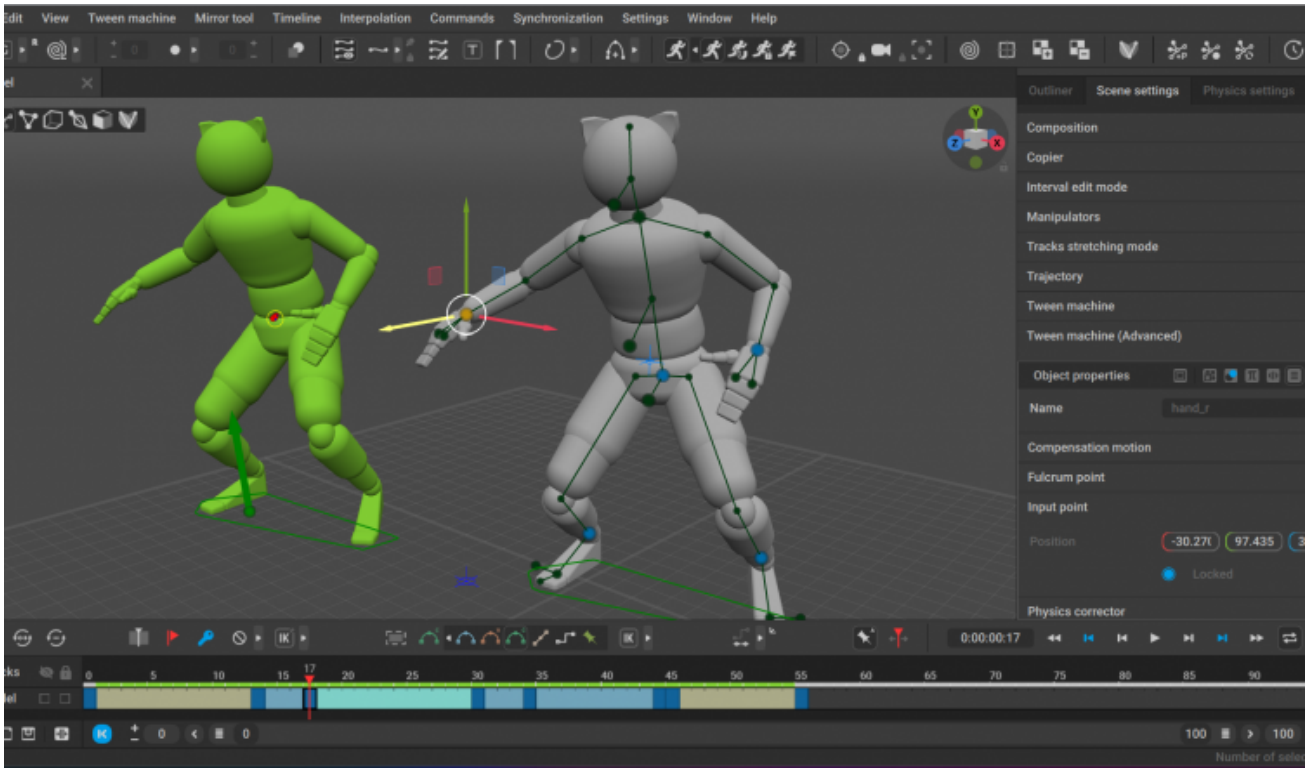

Figure 4: Cascadeur

1.2.2 Justification

The purpose of this research work is to focus on a specific aspect of the animation production process: automating the transition between a 3D model and a 2D model with specific art styles. Some of the previously discussed animation tools allow for motion transfer from 3D to 2D, yet the specific workflow and art styles that this research seeks to improve may not be fully addressed. By developing an automated process that can achieve this transition, this research aims to improve the efficiency and accessibility of the animation production process for artists and studios that may benefit from the usage of this specific workflow.

1.3 Scope

1.3.1 Overall objectives

This thesis has two main objectives. On one hand, is to develop an automated process for Blender that can efficiently convert a 3D model into a 2D model, with a desired art style. Once this had been successfully achieved, the effectiveness and accuracy of the results will be evaluated and compared to those achieved with similar tools. On the other hand, it is also intended to contribute to the advancement of animation technology and the animation industry by improving the animation production process for artists, enthusiasts and studios that may benefit from this specific workflow.

Sub-Objectives

- Review the existing techniques and tools for converting 3D models into 2D art styles
- Identify this technique's strengths and limitations.
- Implement an algorithm that can extract key features from a 3D model and convert them into a 2D representation with a desired art style.
- Evaluate the effectiveness and accuracy of the results
- Compare the results with those achieved through manual conversion as well as other similar tools.
- Optimize the algorithm to ensure that it can handle a wide range of 3D models with varying levels of complexity and detail.

- Develop a user-friendly interface for the tool.
- Conduct user testing to gather feedback on the effectiveness and usability of the automated process.

1.3.2 Requirements

The following requirements are set in place in order to maintain the quality of the thesis.

- Ensure the compatibility of the tool with as many versions of Blender as possible, to increase its reach and usage.
- Prioritize the plugin's usability, to reduce the learning curve for users.
- Keep its resource consumption low and avoid having a significant impact on Blender's performance or slowing down the animation production process.
- Allow for a certain degree of customizability, by including the ability to adjust parameters such as the level of detail, the art style, and other settings related to the conversion process.
- Optimize the output quality of the 2D animations to accurately represent the original 3D model and its motion.
- Prioritize code readability and avoid code complexity.

1.3.3 Possible obstacles and risks

This thesis will potentially meet some risks and obstacles along the way, and these must be taken into account.

- **Technical expertise:** Since I am new to Python scripting with Blender, the first and main obstacles I may face are the challenges with the technical aspects of developing a tool for Blender. Investing time in learning the basics of Python scripting for Blender will be crucial to avoid most of the technical roadblocks later on.
- **Compatibility** Blender is updated quite regularly, and updates may introduce compatibility issues with the plugin. This risk is harder to prevent, but by enforcing good programming practices in the code it can be ensured that even if a Blender update stops being compatible with the plugin, fixing and adapting the plugin will be a less complex task.
- **Performance:** This is an issue that may occur with larger 3D models or scenes since more computational power will be required. Nonetheless, this is something that can be controlled as well by maintaining good programming practices.
- **Product satisfaction:** The resulting 2D rendered animation achieved by this tool will have to be accurate enough to be considered by the stakeholders as a reasonably better alternative to the current more manual processes available, as well as the other similar tools.

1.4 Methodology and rigor

An agile methodology will be enforced for the proper development of this thesis. In this section, the most important aspects of this kind of methodologies, such as regular meetings, dedicated hours of work and progress tracking and how they will be implemented for this research will be discussed.

1.4.1 Methodology

- **Biweekly meetings with the tutor:** There will serve as a way to update them on the progress of the project, discuss any issues or concerns and receive feedback on the work done so far. This allows for any necessary adjustments to be made early on in the project and to ensure that the work is progressing in the right direction.
- **Dedicated work hours:** A scheduled set of weekly hours of dedicated work on the thesis will help ensure that progress is being made consistently and efficiently. By setting aside specific hours each week for this project, it becomes a priority and is less likely to be neglected or pushed aside by other tasks.
- **Workflow:** To keep track of the advancements made, a defined workflow has been created, which includes a list of specific tasks and goals. The tool selected for this specific aspect has been Linear[13]. This will ensure that progress is measurable and allows for easier monitoring of the overall progress of the project.

- **Development Management:** GitHub[16], a web-based hosting service for version control, will be used to manage the development of this thesis. All project files will be stored in a private repository, allowing for easy version control. The primary development environment for the thesis will be Blender's scripting view, with the occasional use of Visual Studio Code. Regular commits will be made to the GitHub repository to track progress and changes made to the project, which will be mainly scripted in Python.

2 Project Planning

During the Thesis Management stage, the current section's goal was to plan how the 540 hours would be allocated to make this thesis. From those 540, 90 hours were destined for the GEP course itself, and the rest were distributed for the thesis implementation itself. However, this section now aims to evaluate the current state of the project and to make an assessment of how much on schedule the execution is actually going. According to the information provided by the faculty, this thesis is worth 18 ECTS, at 25 to 30 hours per credit. Out of these 18, 3 correspond to the GEP module, and the remaining 15, are to be spent on the thesis itself. Taking these details into account, that leaves about 90 hours for completing the GEP, and 450 for the thesis. That includes the development of the project, as well as documenting it and the final lecture.

The following section will be dedicated to outlining how the 480 hours will be allocated and utilized in order to successfully complete the thesis. The different phases of the project will be described, along with their respective resources and requirements. Supplementary tables and figures will be added in order to facilitate the visualisation of the final task arrangement.

2.1 Task Description

It is essential that, before going over the allocation of each task, these are properly defined. Additionally, it should also be kept in mind that, although this initial definition, as well as the resulting schedule, might endure changes as the thesis moves forward, a ground base will assure the correct conditioning of the project.

Task 1. Project Management

This task encompasses the GEP module. Hence, it will be responsible for the planning of the overall project. This task can be structured very easily, based on the four deliverable tasks on Atenea.

- **Task 1.1 - Context and Scope.** Establish clear objectives and justify the project's capabilities. Particularly, emphasizing the identification of the relevant context and concepts pertaining to the thesis.
- **Task 1.2 - Time Planning.** Break down the project into smaller tasks, set time frames for each task, and determine required resources and risk management strategies.
- **Task 1.3 - Budget and Sustainability.** Analyze the economic and sustainable costs associated with pursuing this project.
- **Task 1.4 - Integration of the Definitive Document.** Review and address any feedback provided by the assigned GEP tutor before submitting the final version.

Task 2. Review existing techniques

The initial stage of the project involves analyzing the prevailing techniques and tools utilized to convert 3D models into 2D images rendered with diverse art styles. Our aim is to study the methodology of the most commonly used techniques and evaluate their respective strengths and limitations. This preliminary research is critical for the latter stage of the project, where we will employ the knowledge gathered during this stage to compare the identified tools with the ones that will be developed and implemented for this thesis.

- **Task 2.1 - Literature Review.** Conduct a literature review of existing techniques and tools used to convert 3D models into 2D images with different art styles.
- **Task 2.2 - Identify common techniques and tools.** Identify the most common techniques and tools used in the industry or within the concerning fields.
- **Task 2.3 - Gather data on methodology.** Gather data on the methodology of each technique or tool, including the algorithms and processes involved.
- **Task 2.4 - Analyze strengths and limitations.** Analyze the strengths and limitations of each one, with a focus on factors such as accuracy, speed, complexity, and compatibility with different types of 3D models and art styles.
- **Task 2.5 - Compile a comparative report.** Organize and summarize the findings in order to use them for comparison with the tool that will be developed for the thesis.

Task 3. Development and Implementation of the Blender Tool

After obtaining a comprehensive understanding of the current state of affairs, the next step is to create and implement a Blender tool that can render 3D models as 2D images using various art styles, including Pixel Art. This phase of the project will entail a broad range of tasks, from gaining a thorough understanding of Python plugin development in Blender to creating a usable tool that meets the project's requirements.

- **Task 3.1 - Define requirements.** Identify the desired features and functions of the Blender tool.
- **Task 3.2 - Research Python plugin development.** Investigate the technical aspects of developing plugins in Blender using Python.
- **Task 3.3 - Create design plan.** Develop a detailed plan or blueprint for the Blender tool based on the requirements established.
- **Task 3.4 - Implementation of the core functionality.** Write the code for the core functionality of the Blender tool.
- **Task 3.5 - Implement diverse art styles.** Incorporate different art styles into the Blender tool's functionality.
- **Task 3.6 - Test and refine.** Test the performance of the Blender tool, identify bugs or issues, and make necessary refinements.
- **Task 3.7 - Produce a usable tool.** Finalize the Blender tool and ensure it is user-friendly and meets the requirements set at the beginning of the project.

Task 4. Compare the Blender tool with existing techniques.

In this task, the produced Blender tool which was developed and implemented in the previous stage will be compared with the existing tools. The goal is to evaluate the strengths and weaknesses of our plugin in comparison to the other techniques. This evaluation will help determine the effectiveness and efficiency of the Blender tool, as well as identify any areas for improvement. The results of this task will be used to draw conclusions about the usefulness and potential of our tool for rendering 3D models as 2D images with different art styles, in comparison to the others.

- **Task 4.1 - Define criteria for comparison.** Establish a set of criteria for evaluating the Blender tool and the existing techniques. It will include metrics that quantify aspects such as quality of output, ease of use, processing time, etc.
- **Task 4.2 - Compare Blender tool with existing techniques.** Use the criteria established in the previous task to compare the Blender tool with the existing techniques. Evaluate the strengths and weaknesses of the Blender tool in comparison to the other techniques, and identify any areas for improvement.
- **Task 4.3 - Conclusions.** Analyze the results of the comparison and draw conclusions about the usefulness and potential of our Blender tool for rendering 3D models as 2D images with different art styles, in comparison to the other techniques.

Task 5. Documentation and Presentation

This task involves consistent documentation of the implementation process after each step. The goal is to have a record of the project's progress and to aid in the creation of the final thesis write-up. Additionally, a concluding presentation summarizing the concepts will be prepared for the final lecture.

2.2 Resources

To ensure efficient planning of the thesis, it's crucial to take into account the necessary resources needed for its execution. These resources can be classified into four groups, namely human, material, software, and hardware, which we will discuss in this section.

- **R1. Hardware Resources.** In this case, the only hardware resource will be the computer on which the thesis will be developed. That is a MacBook Pro M1 from 2020 with 512 GB of memory and 16 GB of RAM.
- **R2. Human Resources.** The primary human resources allocated for this project comprise the designated researcher, Beatriz Gomes da Costa, and this thesis tutor, Rubèn Tous Liesa.

- **R3. Material Resources.** This encompasses all the necessary documentation to facilitate familiarity with Python scripting in Blender, as well as comprehending the technology underlying the existing tools to be scrutinized. Below there's a preliminary list with the materials we anticipate consulting.
  - **R3.1** Blender documentation and tutorials
  - **R3.2** Python scripting documentation and tutorials
  - **R3.3** Documentation and technical specifications of existing style transfer tools
  - **R3.4** Relevant academic literature on style transfer and related topics
- **R4. Software Resources.**
  - **R4.1** Blender for developing and implementing the plugin
  - **R4.2** Python as selected language for scripting the plugin
  - **R4.3** Visual Studio Code as the selected IDE
  - **R4.4** LaTeX for documenting the thesis
  - **R4.5** Google Sheets for analyzing and organizing the data.

2.2.1 Summary of the Tasks

The following table provides a summary of the aforementioned tasks, including their anticipated dependencies and an approximate timeline.

ID	Task	Time (h)	Dependencies	Resources
1	**Project Management**	**80**	–	R1, R2
1.1	Context and Scope	20	–	–
1.2	Time Planning	10	1.1	–
1.3	Budget and Sustainability	15	1.2	–
1.4	Integration of Definitive Document	35	1.3	–
2	**Review**	**100**	–	R3.4
2.1	Literature Review	20	–	R3.4
2.2	Identify Common Techniques and Tools	20	2.1	R3
2.3	Gather Data on Methodology	20	2.2	R3
2.4	Analyze Strengths and Limitations	30	2.3	–
2.5	Compile a Comparative Report	10	2.4	–
3	**Implement**	**150**	2.5	R1, R3, R4
3.1	Define Requirements	10	–	–
3.2	Research Python Plugin Development	40	3.1	R3.2
3.3	Create Design Plan	20	3.2	–
3.4	Implementation of Core Functionality	40	3.3	R4.1, R4.2
3.5	Implement Diverse Art Styles	20	3.4	R3, R4
3.6	Test and Refine	20	3.5	R4.5
3.7	Produce a Usable Tool	10	3.6	–
4	**Compare**	**30**	3.7	R3.3, R4.3
4.1	Define Criteria for Comparison	10	–	–
4.2	Compare Blender Tool with Existing Techniques	15	4.1	R3.3, R4.3
4.3	Conclusions	5	4.2	–
5	**Documentation and Presentation**	**10+**	4.3	R4.4
5.1	Consistent Documentation	–	–	R4.4
5.2	Concluding Presentation	10	5.1	–

Table 1: Tasks summary - duration and dependencies

2.3 Gantt Chart

The Gantt Chart below represents the proposed schedule for completing the project. The time frame allows for ample margin to allocate the expected hours for each task, and the project is expected to be completed slightly ahead of the official due date due to this reason.

Given that the size of the image makes it quite hard to read its contents, a full-sized version will be found in **Annex A**.

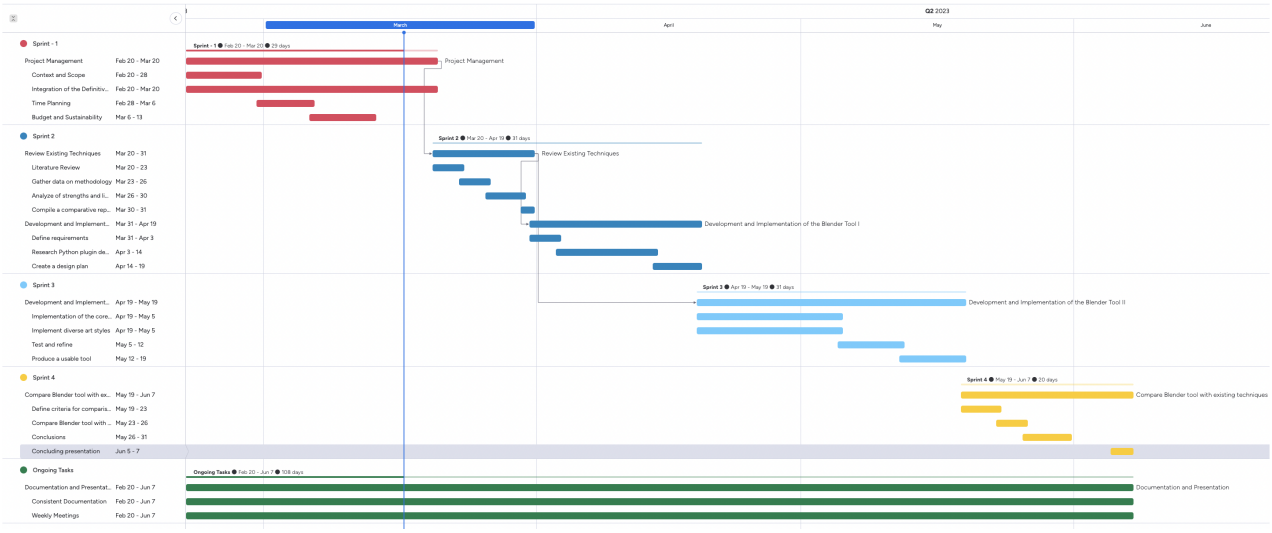

Figure 5: Planned Tasks - Gantt Chart

2.4 Risk Management

It is important to take into account how we will handle any potential risks that may arise during the course of the project. These risks and challenges have been previously identified, and this section will explore possible alternatives and solutions to overcome them as they occur.

2.4.1 Technical expertise

Since I am new to Python scripting with Blender, the first and main obstacles I may face are the challenges with the technical aspects of developing a tool for Blender.

Facing technical difficulties in developing the Blender tool can lead to project delays and even failure to meet the project's deadlines. To prevent this, it is crucial to invest time in learning the fundamentals of Python scripting for Blender before starting the project. Additionally, seeking help from online communities and resources like forums, documentation, and tutorials to address specific technical challenges is important. The project plan already includes tasks dedicated to familiarizing oneself with the technical tools that will be used.

2.4.2 Compatibility

The project may face compatibility issues with Blender updates since Blender is frequently updated. The impact of compatibility issues caused by Blender updates can be significant, as it can potentially render the plugin unusable and require considerable effort to update it. To overcome this risk, the code will be designed and implemented following good programming practices to ensure that it is modular, adaptable, and easy to maintain. Additionally, I will make an effort to stay up-to-date with Blender updates and proactively address any compatibility issues that may arise.

2.4.3 Performance

One potential challenge that may arise in the project is the need for greater computational power when working with larger 3D models or scenes.

The impact of this issue on the project can be delays or slower performance when working with larger 3D models or scenes. To overcome this, it is important to, once again, maintain good programming practices, such as optimizing the code and minimizing or eliminating unnecessary calculations. Additionally, testing the tool with various sizes of models and scenes beforehand will help identify any potential performance issues early on, allowing for adjustments to be made.

2.4.4 Project Satisfaction

The precision of the 2D animation produced by this tool must meet a certain standard to be deemed as a viable replacement for the current manual processes and other similar tools by the stakeholders.

If the resulting 2D-rendered animation is not accurate enough, it may not be considered a viable alternative, leading to the potential rejection of the tool and wasted resources. To overcome this, we will thoroughly test and refine the accuracy of the tool before presenting it to the public. This can be achieved by validating the output against known accurate results, seeking feedback from experts in the field, and continuously improving the algorithms and techniques used in the tool.

2.5 Contingency Plan

In the event of unexpected delays caused by any of the risks identified in the previous section, the first step will be to evaluate the situation and determine the extent of the impact on the project's schedule. If the delay is minor, extra work hours can be put in to make up for the lost time. It is worth noting that the current project plan already includes additional hours that can be utilized in case of a delay. However, if the delay is significant and cannot be resolved through additional work hours, reducing the scope of the project may be necessary.

Reducing the scope of the project will involve dropping functionalities from the final product. This will allow the prioritization of the most essential parts of the project, and ensure that they are completed on time. It's important to note that reducing the scope of the project will be considered a last resort. Before resorting to this option, all other measures, such as seeking help from online resources or consulting with experts in the field, will be explored.

3 Budget

Next up in the project preparation process, is determining how much it'll cost to complete this project. We'll break this down into two parts: human resources costs and material expenses. Additionally, the possible changes to the budget will be considered and we'll come up with ways to keep things under control.

3.1 Human Resources Costs

To examine the staff costs, the first step will be to identify the specific roles that are needed to perform the different tasks outlined in the project plan. After determining the roles, we will establish an average salary for each role and calculate the total cost by multiplying the hourly wage by the estimated number of hours required for each task.

There will be two people taking up different roles. On one hand, the director will take up the roles of both Team Lead **(PL)** and Engineering Manager **(EM)**. On the other, as the author of this project, I will take charge of the following roles:

- Technical Project Manager **(PM)** for establishing the scope and context of the project, as well as defining a time planning schedule, providing a budget, and a sustainability assessment.
- Researcher **(R)** - For conducting research to identify existing tools and processes, developing hypotheses, and testing them.
- Technical Writer **(TW)** - For producing the thesis itself, as well as other documentation necessary for the project.
- Software Developer **(SD)** - For developing code to build the tool, supervised by the director.
- Tester **(T)** - For testing the tool and ensuring that it meets the necessary standards and specifications.
- Data Analyst **(DA)** - For collecting, analyzing and interpreting the data regarding the different tools that will be compared.

The following table displays the hourly costs of various job titles, with salary information obtained from two websites ([14] & [15]). The chosen roles were those which were relevant to the current project and could be found on both websites. When the same role appeared on both websites, we used the data from [22], as it appeared to be more recent.

To ensure accuracy for the personnel involved in this project, all salary information has been based on means in Spain, and all roles have been assumed to be at entry-level. The social security cost has been estimated as 30% of the base gross salary, and a full-time salary has been assumed. This means that the total salary is divided by 40 (hours/week) multiplied by 52 working weeks, resulting in a total of 2080 hours of work per year.

Role	**Annual Salary (€)**	**Social Security cost (€)**	**Cost (€/h)**
Project Lead	42,014	12,604	26.25
Engineering Manager	38,280	11,484	23.92
Technical Project Manager	61,432	18,429	38.39
Researcher	51,416	15,424	32.13
Technical Writer	31,545	9,463	19.71
Software Developer	38,012	11,403	23.76
Tester	29,368	8,810	18.35
Data Analyst	38,592	11,577	24.12

Table 2: Estimated Salary per Role, based in Spain

Once the costs for each role were determined, we allocated the defined tasks to the predicted roles and assigned the appropriate number of hours per role.

Task	Hours	PL	EM	PM	R	TW	SD	T	DA
Project Management	80	0	0	80	0	0	0	0	0
Context and Scope	20	0	0	20	0	0	0	0	0
Time Planning	10	0	0	10	0	0	0	0	0
Budget and Sustainability	15	0	0	15	0	0	0	0	0
Integration of Definitive Document	35	0	0	35	0	0	0	0	0
Review	100	0	0	0	100	0	0	0	0
Literature Review	20	0	0	0	20	0	0	0	0
Identify Common Techniques and Tools	20	0	0	0	20	0	0	0	0
Gather Data on Methodology	20	0	0	0	20	0	0	0	0
Analyze Strengths and Limitations	30	0	0	0	30	0	0	0	0
Compile a Comparative Report	10	0	0	0	10	0	0	0	0
Implement	150	0	0	30	40	0	90	20	0
Define Requirements	10	0	0	10	0	0	0	0	0
Research Python Plugin Development	40	0	0	0	40	0	0	0	0
Create a Design Plan	20	0	0	20	0	0	0	0	0
Implementation of Core Functionality	40	0	0	0	0	0	40	0	0
Implement Diverse Art Styles	20	0	0	0	0	0	20	0	0
Test and Refine	20	0	0	0	0	0	20	20	0
Produce a Usable Tool	10	0	0	0	0	0	10	0	0
Compare	30	0	0	0	0	0	0	0	30
Define Criteria for Comparison	10	0	0	0	0	0	0	0	10
Compare Blender Tool with Existing Techniques	15	0	0	0	0	0	0	0	15
Conclusions	5	0	0	0	0	0	0	0	5
Documentation and Presentation	30+	20	20	20	0	+	0	0	0
Consistent Documentation	+	0	0	0	0	+	0	0	0
Weekly Meetings	20	20	20	20	0	0	0	0	0
Concluding Presentation	10	0	0	0	0	0	0	0	0
Total Hours per Role	-	20	20	130	140	+	90	20	30
Total (€)	-	525	478.4	4,990.7	4,498.2	-	2,138.4	367	723,6

Table 3: Human Resources Task Division and Total Cost

Based on the breakdown of the project tasks, the estimated total cost for staff salaries is approximately **13,721.3€**.

3.2 Material Expenses

In this section, we will address costs that are not related to staff, such as expenses for software, hardware, and other miscellaneous costs.

3.2.1 Hardware Amortization

As stated in the Resources section, the MacBook Pro M1 is the only required hardware for this thesis and project, and it is expected to be used for approximately 540 hours. The MacBook Pro was purchased for 1300 euros in December 2020, and assuming that its useful life is 4 years (or 35,040 hours), the proportion of the useful life used during the thesis period can be estimated to be approximately 1.54%. Based on this proportion, the amortization cost for the MacBook Pro during the thesis period can be estimated to be around 5.00 euros.

However, this is still only an estimate; the actual amortization cost may differ depending on various factors such as maintenance and repair costs, changes in technology, and other unforeseen circumstances.

3.2.2 Software

There is no cost associated with the software resources required for the development of this project, as all the necessary software is available for free. Therefore, the total cost for this section is zero.

3.2.3 Indirect Costs

Electricity is our main indirect cost. Currently, the average cost of electricity is around 0.10408 euros per kilowatt-hour. The laptop we're using for this thesis consumes around 90 watts of power per hour on average. Based on these figures, the total cost of electricity for using the laptop for 540 hours would be of 50€.

3.3 Budget Deviation

3.3.1 Contingency

To account for potential budget deviations, we will allocate contingency budgets for each cost section. We will assume a 15% contingency budget for all sections, which is an average percentage recommended for this sector based on the documentation provided in the GEP course. This means that we will set aside an additional 15% of the estimated costs for each section to cover unexpected expenses or deviations from the original plan. This will help us ensure that we have enough resources to complete the thesis even if we encounter unforeseen costs or challenges.

Cost	**Contingency Budget (€)**
Human Resources	2,058.2
Hardware Amortization	0.75
Indirect Costs	7.5
Total	**2,066.45**

Table 4: Contingency Budget

3.4 Incidental Costs

In this section, we will estimate the potential monetary impact of the risks we have identified in section 2. Only two of the risks we identified, namely **Technical Expertise** and **Compatibility**, are likely to have direct consequences on the budget, as they may require additional hours and resources to resolve. However, we also acknowledge that hardware repairs could potentially cause significant incidental costs if not addressed properly. Therefore, we will include this as a possible risk in our estimation.

It's important to note that the likelihood and severity of these risks may differ:

- **Technical Expertise:** Given my limited experience in the field, Technical Expertise is very likely to happen. I have limited knowledge and experience in some of the technical areas related to this project, such as scripting in Python for Blender plugins, which increases the likelihood of technical difficulties. As has been stated in section two, this may require additional hours and resources to resolve.
- **Hardware Breakdowns:** When it comes to hardware breakdowns, no significant issues have arisen so far with the computer I will be using for the development of this project. Therefore, while hardware repairs have been included as a possible risk in our estimation, it is less likely to occur.
- **Compatibility:** While minor compatibility issues are expected to arise, they are not likely to affect the overall completion of the project. The software and tools used in this project may require specific versions, updates, or dependencies that could become obsolete in the future. However, steps to ensure that, at least for the time being, the software and tools used in the project are compatible with each other, will be taken. Therefore, although the likelihood of this risk is quite high, it won't have an immediate impact on the project's timeline.

In the following table, we have outlined the roles responsible for addressing these potential roadblocks, as well as the estimated costs associated with each risk. We hope that this estimation will allow us to be better prepared for any potential budget deviations that may arise, and help ensure the successful completion of this thesis project.

3.5 Final Budget

Below is a brief overview of the total budget allocated for the project:

Risk	Hours	Role(s)	Total(€)
Technical Expertise	20	R & SD	558.9
Compatibility	10	SD & T	210.6
Hardware breakdown	-	-	150
Total	30	-	919,5

Table 5: Risk Management Costs

Cost	Total (€)
Human Resources	13,721.3
Hardware Amortization	5
Software	0
Indirect Costs	50
Contingency Budget	1,637
Incidental Costs	919.5
Total	**15,332.8**

Table 6: Total Budget

3.6 Management Control

Effective budget management is crucial to the success of any project. To manage a budget effectively, it is important to track both the expected costs and the actual costs of each task. This can be done by regularly reviewing and updating the budget as the project progresses.

The method we will use for managing the budget is to create a detailed plan that outlines all of the expected costs for each task, including staffing, resources, etc. This plan will also include the provided contingency budget to account for any unexpected costs that may arise during the project.

As the project progresses, we will track the actual costs of each task and compare them to the expected costs. To do so, records of every expense will be kept. By regularly reviewing the actual costs and comparing them to the expected costs, adjustments can be made to the budget as needed to ensure that the project stays on track financially.

Additionally, we will make sure to prioritize spending based on the most critical tasks and objectives of the project, even though a specific primary budget has already been set aside for each section. That is because, by doing so, we can help to ensure that resources are allocated effectively and that the most important tasks are completed on time and within budget.

4 Sustainability

4.1 Survey's Self-Assessment and insights

In reflecting on the sustainability survey I recently took, I came to the realization that I had not been fully aware of the impact that software engineering projects can have on the environment. Prior to taking the survey, my understanding of sustainability was limited to creating tangible products that were zero waste. I had not considered the concept of sustainable software development in-depth, especially in terms of metrics for calculating their impact, or project viability.

Throughout my Computer Engineering studies, we did focus on some of the economic aspects of developing products within our field. I am aware of the existence of an optional subject that goes over the environmental aspects, but unfortunately, I haven't had a chance to take this course. As a result, I had not been exposed to many of the concepts discussed in the survey.

I identified several gaps in my understanding, especially in terms of metrics and calculating their impact. Moving forward, I plan to seek out more information and resources to increase my understanding of sustainable software development.

4.2 Economic Dimension

PPP - Reflection on the project's estimated cost

I have made efforts to consider the various factors that may impact the project's cost, such as its scope, required resources, the complexity of the development process, and potential risks. By taking these factors into account and estimating the maximum expected time for the project, I believe I have created a quite comprehensive cost estimation that should cover the entirety of the project while falling within an acceptable range for a project in this sector.

Lifespan

Regarding the state of the art, there are various other techniques and alternatives available that are relevant to the plugin being developed in this thesis. For instance, there are other plugins such as Toonkit or the Grease Pencil tool within Blender that can be used for 2D animation. Additionally, there are also standalone software applications such as Anime Studio and Toon Boom Harmony that specialize in 2D animation. These will be discussed further in the future.

However, the plugin we aim to create is unique in its specific focus on automating the process of creating 2D animations from 3D models with various art styles. This means that the lifespan of the product is not directly affected by the existing alternatives, given that it uses a different approach and has a specific focus. Nevertheless, the tool needs to be of high quality to be considered a viable alternative to existing techniques.

The potential to adapt the plugin to include additional art styles can also significantly impact the lifespan of the product. By expanding the plugin's capabilities to support a wider range of art styles, it can attract a larger audience and user base. This can lead to continued interest and development in the plugin over time, which can ultimately result in a longer lifespan for the product. Additionally, the ability to make minor adjustments to the plugin in response to periodic updates to Blender can also contribute to its longevity.

By taking into account the various techniques and alternatives available in the state of the art and focusing on the unique aspects and potential for expansion of the proposed plugin, it is quite reasonable to expect a long lifespan for the product.

Economic Improvements

The solution being developed in this thesis aims to improve the economic aspect of the animation process by reducing the time and effort required to create 2D animations from 3D models with different art styles. The existing techniques for this process often require significant manual effort and can be time-consuming, leading to higher costs in terms of labour and resources. Moreover, some of these existing techniques are not free and require a financial investment to use them, which adds up to the overall costs of the animation process.

4.3 Environmental Dimension

PPP - Reflection on the Project's Environmental Impact and resource reuse

While it's true that we haven't performed a thorough report regarding sustainability, based on the aspects of the project, the estimated environmental impact is likely to be minimal. As we know, the project involves developing an automated process for Blender, which will be developed on a single computer, the energy consumption associated with the project is likely to be relatively low, as we have seen in the previous section. Furthermore, Blender is open-source software and so will be the plugin. This can contribute positively to the environmental impact by reducing the amount of proprietary software that needs to be used or produced.

Regarding the code, and as per good practices, the aim is to make it as efficient as possible. By doing so, we will help reduce the amount of energy required to run the plugin. Additionally, Blender is already an efficient open-source software, which makes it likely to further minimize the environmental impact.

Lifespan

In terms of environmental impact, the solution being developed in this thesis has the potential to improve upon the existing techniques for creating 2D animations from 3D models. By automating the process and reducing the need for manual labour, the plugin can help to reduce the carbon footprint associated with the animation production process.

While there are several state-of-the-art techniques for converting 3D models into 2D animations using various methods, they are not without their environmental issues. For example, some of the existing techniques may require multiple iterations of rendering and manual adjustments, leading to longer rendering times and higher energy consumption.

In addition, the plugin is being developed for use within Blender, which is an open-source software program. This means that the plugin will also be open source, allowing for greater collaboration and potential contributions from the community. Open-source development can lead to more efficient and sustainable software development practices, as it allows for a wider range of perspectives and contributions, which can help to reduce development time and minimize errors.

4.4 Social Dimension

PPP - Personal growth

Firstly, as a developer, I will have the opportunity to expand my technical skills and knowledge of programming and scripting in Python, as well as gain a deeper understanding of the software development process. I will be able to apply my technical skills to create a complex tool that can automate a specific process and improve the workflow of 2D animators. Overall, this thesis and project can help me improve my programming and problem-solving skills; both are very valuable within my fields of interest, in a liberal context.

Additionally, this project will also help me develop essential soft skills such as time and project management, as well as communication. I will have to manage the available time efficiently to meet the project's deadlines and manage the project by breaking it down into smaller tasks and tracking progress. Some of them have already been put into practice thanks to the GEP course.

And last, but definitely not least, working on a project that contributes to the advancement of animation technology can be a rewarding experience in and of itself. Knowing that my thesis can make a difference in the animation industry by improving the workflow for artists, studios and other stakeholders can be a source of great satisfaction.

Lifespan

Currently, the process of converting 3D models into 2D models with a desired art style for animation is often a time-consuming and manual process. Many artists and studios use a combination of software tools and techniques currently available, in order to achieve the desired results. While these plugins and tools aim to automate parts of the process, the process is still quite consuming in many aspects.

The solution being developed in this thesis aims to improve the animation production process by providing an automated tool that simplifies and accelerates the animation process, reducing the overall time and effort required. By doing so, the plugin can improve the quality of life for artists and studios, allowing them to focus on the creative aspects of their work and reduce the manual labour involved in the process.

The need for this project is almost evident in the animation industry, as we have seen how the time-consuming and manual workflows can lead to higher costs in terms of labour and resources, as well as potential delays in project timelines. By providing a more efficient and accessible alternative, the plugin can address these challenges and contribute to the advancement of animation technology and the animation industry.

5 Laws And Regulations

In the development and use of software, various laws and legislations come into play. These often revolve around issues of licensing, copyright, patents, data protection, and privacy among others. In the context of this specific project, several of these aspects are relevant and are discussed below.

5.1 License, Copyright, and Patent

I intend to provide this project under the Apache License 2.0. This is a permissive license that is widely used in the open-source community. It allows users to use, distribute, and modify the software. Moreover, unlike some other licenses, it also provides an express grant of patent rights from contributors to users. The choice of the Apache License is intended to encourage contributions to the project from the broader open-source community.

5.2 Other Legal Aspects

Beyond licensing, there are also other legal aspects that should be considered for any project. For instance, when it comes to data protection and privacy laws, this project does not involve the manipulation or storage of user data. Thus, many of the typical concerns in this area are not relevant in this case. Furthermore, the project exists in the context of Blender, serving as a plugin script to extend its capabilities. As such, it inherits the legal regulations and guidelines set out by Blender's own framework and community.

6 Deviations with respect to the Original Planning

Previously, the plan initially established during Project Management (GEP) was described. However, during the implementation of said planning the project was affected by some alterations, which will be described here. This section will go over the unplanned circumstances, how these affected the initial planning, and finally, what was done to overcome them.

6.1 Literature Review

The initial stage of the project involved analyzing the prevailing techniques and tools utilized to convert 3D models into 2D images rendered with diverse art styles. The aim was to study the methodology of the most commonly used techniques and evaluate their respective strengths and limitations.

It was established pretty early on that there were very few tools that accomplished exactly what this project aimed to create. Nevertheless, a thorough review was still accomplished, as I explored other techniques and recent breakthroughs in research that facilitated the 2D animation process in some other way. This wider search revealed some very interesting facts about the current state of the art. For instance, some of the newer techniques use Artificial Intelligence[4] in order to predict a sequence of frames, when given the first few. While most of these techniques are still in early development, the achieved results are remarkable, and we can expect to see great improvements in the near future.

Gathering detailed information about the way existing tools worked proved challenging due to the fact that a vast majority were not open-source. This obstacle required a shift in the approach, focusing less on a deep understanding of alternative methodologies and more on achieving similar results with the proposed innovative approach. Thus, the lack of similar pipelines in the reviewed tools reinforced our path towards a unique tool implementation.

As a result, the research also focused more on studying Blender's scripting extensively, as this platform was selected for developing the plugin. The information obtained during this period consolidated the base for the implementation portion of this thesis. Key learnings about the usage of shader[17] and compositor nodes[18] and Python scripting in Blender were acquired and proven to be crucial later on, as the resulting plugin is mainly based on these concepts.

An unforeseen yet welcomed change to the initial plan was a visit to an animation company named DEVICE[19]. This experience offered insights into real-world practices and industry demands that would have been much harder to obtain from an ordinary literature review. By observing their work, specifically their 2D Pixel Art creations, and understanding how time-consuming their processes could get, the initial stakeholders of the tool were confirmed.

Overall, the insights acquired during this phase did not have a negative impact on the initial planning. Instead, it contributed to consolidating some aspects in a way that the less in-depth exploration done during the initial GEP phase would not have allowed. To begin with, it helped define the scope in more detail by providing a real understanding of the complexity level. Additionally, after ascertaining the lack of tools following our approach, a rethinking of the comparison and evaluation step had to be made. This is because otherwise, certain metrics could not be applicable depending on the method.

6.2 Implementation

During the planning phase of this thesis, potential risks were identified that could affect the progress of the project; especially the implementation phase. As expected, we encountered some of these risks, leading to necessary adjustments in our planning. However, the mitigation strategies proposed earlier helped in overcoming these obstacles and ensuring the thesis development stayed on track. Notably, with the exception of a performance issue that will be discussed below, none of the barriers encountered was too severe. Here, we will revise the risks from **Section 2.4** that ended up taking place, and how these affected the course of the thesis development.

6.2.1 Technical expertise

As expected, my lack of experience with Blender slowed the progress at the beginning. However, thanks to the accommodating initial schedule, only a few extra hours were needed to overcome this particular problem. These extra hours were mostly spent studying the documentation in more depth and watching tutorials that

explained how certain Blender aspects worked. Gradually, I developed a familiarity with the software, which allowed me to make up for the slow start and keep the project's timeline fairly intact.

6.2.2 Compatibility

The project was planned with an awareness of the frequent updates to Blender, which could potentially cause compatibility issues in the future. The impact of these updates could be considerable, potentially rendering the plugin unusable and requiring significant effort for updates. The original plan was designed to tackle this obstacle by sticking to good programming practices, ensuring the code is modular and easy to understand and hence, maintain. This is not an issue I have encountered yet, given that the project is still up to date. Nonetheless, I expect the initial contingency plan to work as well as the others. Besides, even if it did not, as the thesis will already have been finished by then, time would not be a major issue anymore, and I would be able to solve the problem with ease.

6.2.3 Project Satisfaction

The original plan was also based on a need for the tool to meet certain quality standards. This is because the tool strives to be considered an actual viable alternative to the existing processes, which are generally more manual and time-consuming. I faced this issue quite early on, with the initial tests I made using pre-created models from Mixamo's assets. The highly detailed models with built-in textures conflicted with the proposed pipeline and the desired results, thus challenging the plan's premise of accurate rendering. There are some images portraying these early results in **Annex A**. As can be seen there, the problem was caused by the over-detailed textures reacting to light in an undesired way.

Once the problem was identified, it was clear that this meant that the plan needed to include the implementation of an extra feature: the simplification of meshes and splitting of the original model into colour-based sections. This modification would allow users to select simpler colours without excessively detailed textures and successfully render the simplified model. Thereby, this added feature implied aligning the state of these pre-made models with the custom-made ones, and allowing users to have the same level of control over both.

6.2.4 Performance

It was also anticipated to face performance-related challenges, especially when working with larger 3D models or scenes. As predicted, I did encounter a performance issue, that due to limitations in Blender, I was not able to solve as satisfactorily as I would have wanted. Essentially, the problem faced was caused by the newest feature introduced in the previous section. The adaptation of meshes with considerable amounts of polygons made the method in charge of processing take several hours to finish execution. And while I could partially improve these execution times, Blender's inability to support multi-threading[20] rendered the optimization of the code a true challenge. In fact, the slow execution time for this mesh processor method has been the most severe issue encountered during the implementation phase.

6.3 Tool Comparison

After no suitable or accessible tools were found to compare to our add-on, a rethinking of this whole stage of the thesis had to be done. After accepting the fact that a full one-to-one comparison would not be possible, I opted for a feature-based approach. In other words, tools that recreated an art style in a similar way to our tool were selected, and the results were compared visually, as well as by other measures that will be discussed in **Section 11.1**. All in all, a good comparison was achieved, although some of the features were just evaluated, without a counter tool to compare them with.

6.4 Final Scope

During the literature review, some modifications were made to the original planning based on the derived insights. Firstly, the reach of the literature review was enlarged, due to the lack of material in the specific area we were initially searching. This was done in order to include techniques which, while following different methodologies, provided valuable insight into the current state of the art. We will go over the knowledge acquired during this phase on Sections 7 and 8.

Secondly, some alterations and specifications were applied during the implementation step as well. When it comes to the scope, these changes affected the initial requirements and features of the final product. The list below shows the final state of the requirements, after taking into account the deviations from the original planning.

- **Number of explored art styles:** I ended up settling with three main art styles, as I found that a larger number could leave me running behind schedule pretty quickly. The chosen art styles are Pixel Art, Cel Shading and Outlined Cel Shading.
- **Automating:** It is worth mentioning again that this thesis focuses on the final section of a pipeline that relies on more manageable animated 3D models to render 2D animations. The area in question ranges from importing the animated 3D model into Blender to rendering in 2D. This section basically substitutes the whole step of manually drawing every individual frame on a conventional 2D animating process, which can take considerable time, depending on the length of the final animation. Therefore, efficiently automating as much of this process as possible is crucial.
- **Multiple input cases:** As previously mentioned, the tool focuses on frame-by-frame drawings in the traditional 2D animation process. However, considering the proposed pipeline's nature, the input 3D models that users intend to render as 2D animations may originate from various sources. These models could be custom-made, generated using AI with motion capture, or obtained from asset databases like Mixamo. Therefore, to ensure the tool's ease of use across these diverse origins, additional features need to be incorporated. This is necessary because the level of control over the final 3D model varies significantly among these different sources.
- **Efficiency:** For a tool to be successful and preferred over other tools or techniques, it must be more efficient than its alternatives. Thus why in the process of automating this pipeline I made it a priority to keep a low execution time. This has presented some challenges, as some aspects of the plugin implicated processing all the vertices of the 3D model; a number that scales rather quickly.
- **Multiple output options:** The final 2D animation might not always be the end result. That means that while some users may use the plugin to create a 2D animation, some others might be more interested in using the several generated frames to make a video game sprite or use it as a base for something else. That's why the plugin must also be able to generate the output in different formats and allow the user to pick the one that's best for their project. Mainly, the rendered result could be either a set of separated frames, a video or a sprite sheet. Blender already handles the first two, but the third one will need to be implemented.
- **Result Quality:** There is a notable difference between a Pixel Art image and a pixelated one. The tool needs to be as true to conventional Pixel Art as possible. This remains unchanged with respect to the original requirements, but its presence is highly related to the need for the extra processing features we ended up incorporating.

6.5 Final Tasks and Resources

Changing the originally planned tasks was essential in order to adapt to the updated scope, as well as to alleviate the time issues caused by some of the obstacles found. In this case, the modifications needed were few. This is because the original plan already allocated extra hours in the event that we fell behind schedule, and as we mentioned, the only important change made that directly affected the plan was the addition of some mesh processing features. Furthermore, the resources must be revised as well. We initially classified the resources into four groups, namely human, material, software, and hardware. Despite the divergences from what was originally planned, no changes have been made to this initial planning, as no new or unaccounted-for resource was used. However, it is true that some of the material resources were more available and hence, used, than others. And in a similar way to what happened with the tasks, the specifics of some of the material resources were slightly altered in the final version.

- **R1. Hardware Resources.** In this case, the only hardware resource will be the computer on which the thesis will be developed. That is a MacBook Pro M1 from 2020 with 512 GB of memory and 16 GB of RAM.
- **R2. Human Resources.** The primary human resources allocated for this project comprise the designated researcher, Beatriz Gomes da Costa, and this thesis tutor, Rubèn Tous Liesa.
- **R3. Material Resources.** This encompasses all the necessary documentation to facilitate familiarity with Python scripting in Blender, as well as comprehending the technology underlying the existing tools to be scrutinized. Below there's a preliminary list with the materials we anticipate consulting. Here it's worth mentioning that resource R3.3 was rather scarce to find, which is why the research focused more on the other resources as well as a broad understanding of the pipeline behind these technologies, instead of their implementation.

  - **R3.1** Blender documentation and tutorials
  - **R3.2** Python scripting documentation and tutorials
  - **R3.3** Documentation and technical specifications of existing style transfer tools
  - **R3.4** Relevant academic literature on style transfer and related topics

- **R4. Software Resources.**
  - **R4.1** Blender for developing and implementing the plugin
  - **R4.2** Python as selected language for scripting the plugin
  - **R4.3** Visual Studio Code as the selected IDE
  - **R4.4** LaTeX for documenting the thesis
  - **R4.5** Google Sheets for analyzing and organizing the data.

Table 7 shows how the aforementioned events affected the original scheduling.

ID	Task	Time (h)	Dependencies	Resources
1	**Project Management**	**80**	–	R1, R2
1.1	Context and Scope	20	–	–
1.2	Time Planning	10	1.1	–
1.3	Budget and Sustainability	15	1.2	–
1.4	Integration of Definitive Document	35	1.3	–
2	**Review**	**100**	–	R3.4
2.1	Literature Review	20	–	R3.4
2.2	Identify Common Techniques and Tools	20	2.1	R3
2.3	Gather Data on Methodology	20	2.2	R3
2.4	Analyze Strengths and Limitations	30	2.3	–
2.5	Compile a Comparative Report	10	2.4	–
3	**Implement**	**165**	2.5	R1, R3, R4
3.1	Define Requirements	10	–	–
3.2	Research Python Plugin Development	40	3.1	R3.2
3.3	Create Design Plan	20	3.2	–
3.4	Implementation of Core Functionality	40	3.3	R4.1, R4.2
3.5	Implement Diverse Art Styles	20	3.4	R3, R4
3.6	Implement pre/post-processing features	15	3.5	R3, R4
3.6	Test and Refine	20	3.5	R4.5
3.7	Produce a Usable Tool	10	3.6	–
4	**Compare**	**30**	3.7	R3.3, R4.3
4.1	Define Criteria for fair Comparison	10	–	–
4.2	Compare Blender Tool with Existing Techniques	15	4.1	R3.3, R4.3
4.3	Conclusions	5	4.2	–
5	**Documentation and Presentation**	**10+**	4.3	R4.4
5.1	Consistent Documentation	–	–	R4.4
5.2	Concluding Presentation	10	5.1	–

Table 7: Tasks summary - duration and dependencies

In summary, regarding the just new features were added in order to facilitate the management of the 3D model. Additionally, some components that provide the user with accommodating and useful output options, were added as well. Admittedly, these features don't make the plugin's support bigger, but they do make it more reliable and easy to use. Finally, the scope of the project was settled at three different art styles, as well as the added features for mesh processing and other components that will be discussed further in **Section 10**.

6.6 Updated Gantt Chart

The Gantt Chart below represents the revised schedule for completing the project, taking into account the slight delay. Given that the original timeline already allowed for ample margin to allocate the expected hours for each task, and that the project was expected to be completed slightly ahead of the official due date, the applied changes had no severe consequences.

Given that the size of the image makes it quite hard to read its contents, a full-sized version will be found in **Annex B**.

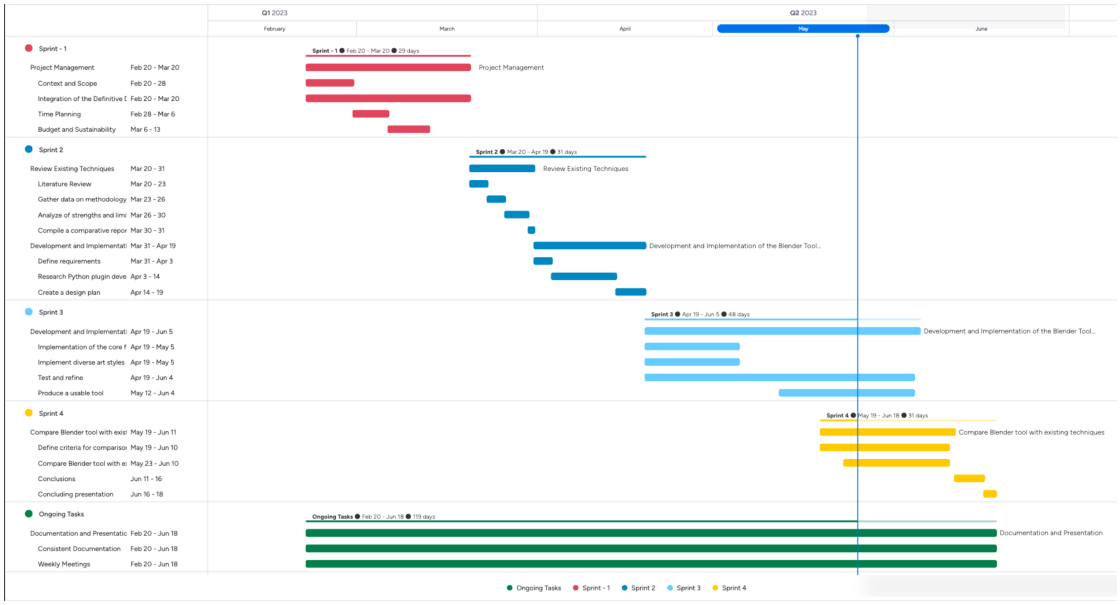

Figure 6: Planned Tasks - Updated Gantt Chart

7 Animation Fundamentals

As the following section explores the current state of the art, it is first important to define general aspects within the concerning context of animation. While this has been already explored in the Project Management (GEP) phase, the current section aims to bring a broader base of fundamentals to the reader. This way, by the end of it, they will be properly equipped to fully understand the context in which the state of the art takes place. Firstly, a quick overview of the evolution of animation will provide an idea of which techniques have been used through the years. Once the overview is complete, a general animation process will be explained, with a special focus on the second stage, as that's where the actual importance lies.

7.1 Evolution of Animation

Animation is a visual art that brings life to images or objects by creating the illusion of motion. This is achieved by sequencing a series of images, where each one is in a slightly different position or frame from the one before. When displayed at a certain speed, our brains process these images as a continuous, moving scene due to a phenomenon known as the persistence of vision[21]. This principle forms the basis of all types of animation, from traditional hand-drawn methods to advanced computer-generated techniques.

There are several types of animation. Each of them has specific characteristics, techniques, and areas of use. Below, we go over them, beginning with the ones that originated the art, until the more complex and cutting-edge techniques.

7.1.1 Traditional Animation

As the name implies, traditional or cel animation is one of the oldest forms of animation. It involves drawing each frame by hand on sheets called cels. Each cel is then photographed. When the series of frames is played back at a fast speed, the images appear to be moving. Originally, the cels were photographed onto a film strip, but nowadays there exist several alternatives.

7.1.2 Rotoscoping

Introduced in 1917 by Max Fleischer, rotoscoping[22] signified a great improvement over the simpler traditional animation. The process involved projecting individual frames of a film onto a transparent panel, for an artist to trace by hand. This was a labour-intensive process, given that each frame was to be traced in this manner. However, given that the artist's focus wasn't on understanding motion anymore, but just on simply tracing their characters over a video frame, the results had a more natural feel to them. Thus, the extra time needed to film and trace single frames was worth it.

Figure 7: Original video

Figure 8: Animation by rotoscoping

7.1.3 Stop Motion

This form of animation involves manipulating real-world objects and photographing them one frame at a time. In essence, stop motion is to modern 3D animation what cel animation and rotoscoping are to two-dimensional animation. Therefore, in a similar fashion, when the series of photographed frames are played back, the object appears to move on its own. There are different types of stop-motion animation, based on the materials of which the figures are made. To mention a few, there is "claymation" (clay figures), puppet animation, and cut-out animation. The Nightmare Before Christmas, for instance, was made using this technique.

7.1.4 Vector-based 2D Animation

In this more modern type of animation, instead of drawing frame by frame, the animator creates the motion by making changes to the properties of a digital object, such as its position, scale, or rotation. These changes are then interpolated over time by the software. Among others, the benefits of using vector-based animation are the ability to scale without compromising image quality, smaller file sizes and, of course, time efficiency. Currently, there are several tools that use this type of animation; Adobe Animate[23] being one of the most used.

7.1.5 3D Animation

This form of animation is created by using computer software to generate three-dimensional images. Essentially, the characters or objects are polygon-based meshes that get rigged with a special object named "skeleton". These objects bind to the mesh and allow animators to control the models, by creating movement frame by frame. This type of animation is very common in animated 3D films and video games. Some Pixar movies exhibit this technique.

7.2 The Animation Process

The animation process refers to the succession of actions that animators follow to take an animated sequence from concept to final completion. These workflows are structured around a series of stages, each with its distinct tasks and objectives. The exact workflow varies depending on the project, the style of animation, the software, and the animation technique used. Nonetheless, most follow the fundamental progression of pre-production, production, and post-production.

Upon completing these stages, the final product is typically a sequence of frames that make up the animation or a video. Ultimately, the format will depend on the purpose the animation is accomplishing. In other words, the final animation could directly be a standalone project, or it could be integrated into a larger project, such as a film or video game. The specific use of the output depends on the objectives defined during the pre-production stage.

7.2.1 The Pre-production Stage

The Pre-production focuses on planning and preparation. It begins with conceptualization, which is where the story or idea for the animation is developed. Then, storyboarding follows. Essentially, this step serves as a way of outlining the sequence of events, in the same way outlining text structures what wants to be said. Additionally, the different characters and the environment designs are created at this stage. When it comes to two-dimensional animation, animatics (namely initial unpolished versions of the animation) are also drawn in this stage to provide a preliminary view of the final product.

7.2.2 The Production Stage

The Production stage is where the animation actually comes to be. It starts with the creation of keyframes, which define the start and end points of the motion sequence. Once again, the way in which these frames are generated depends on the used technique, and we mentioned some of them in the previous section. It's worth noting that some techniques create the outlined frames first, and then the rest are generated in a process known as "inbetweening" or "tweeting". This stage also includes the addition of colour and shading, if there are any to be added.

Considering the various animation techniques we previously covered, it's not hard to realise how complex it can become to manage vast amounts of frames to achieve a realistic result. While newer techniques are more time efficient, the process is still lengthy when it comes to big projects. As a result, we will delve deeper into the strategies employed to tackle large quantities of frames. This will provide a closer look at the workflow of animation production, going from traditional methods to modern automated technologies such as motion

capture and artificial intelligence.

Straight ahead

Straight ahead animation is a elemental approach that consists in drawing frames one by one. Introduced by Ollie Johnson and Frank Thomas in the 80's book **"The Illusion of Life"** [24]. While the unplanned nature of this technique can lead to a natural and fluid series of actions, the lack of planning can also present challenges such as unintentional scaling of the character and lengthy scenes given that the animator has no guide to the target destination in the action. Admittedly, this non-deterministic aspect of the straight ahead strategy can be desired in some cases, for artistic reasons.

Pose to pose

The pose to pose approach puts a solution to the "issues" presented by the previous technique. To ensure a more structured result, the animator first creates the main poses, namely "key" poses, and then fills in the sequence with the in-between frames. As a result, an overall view of the final result is obtained almost at the beginning of the process, which allows the artist to have better control over important aspects such as the total number of frames and the timing. Another advantage over straight ahead animation is that alterations can be implemented with much ease.

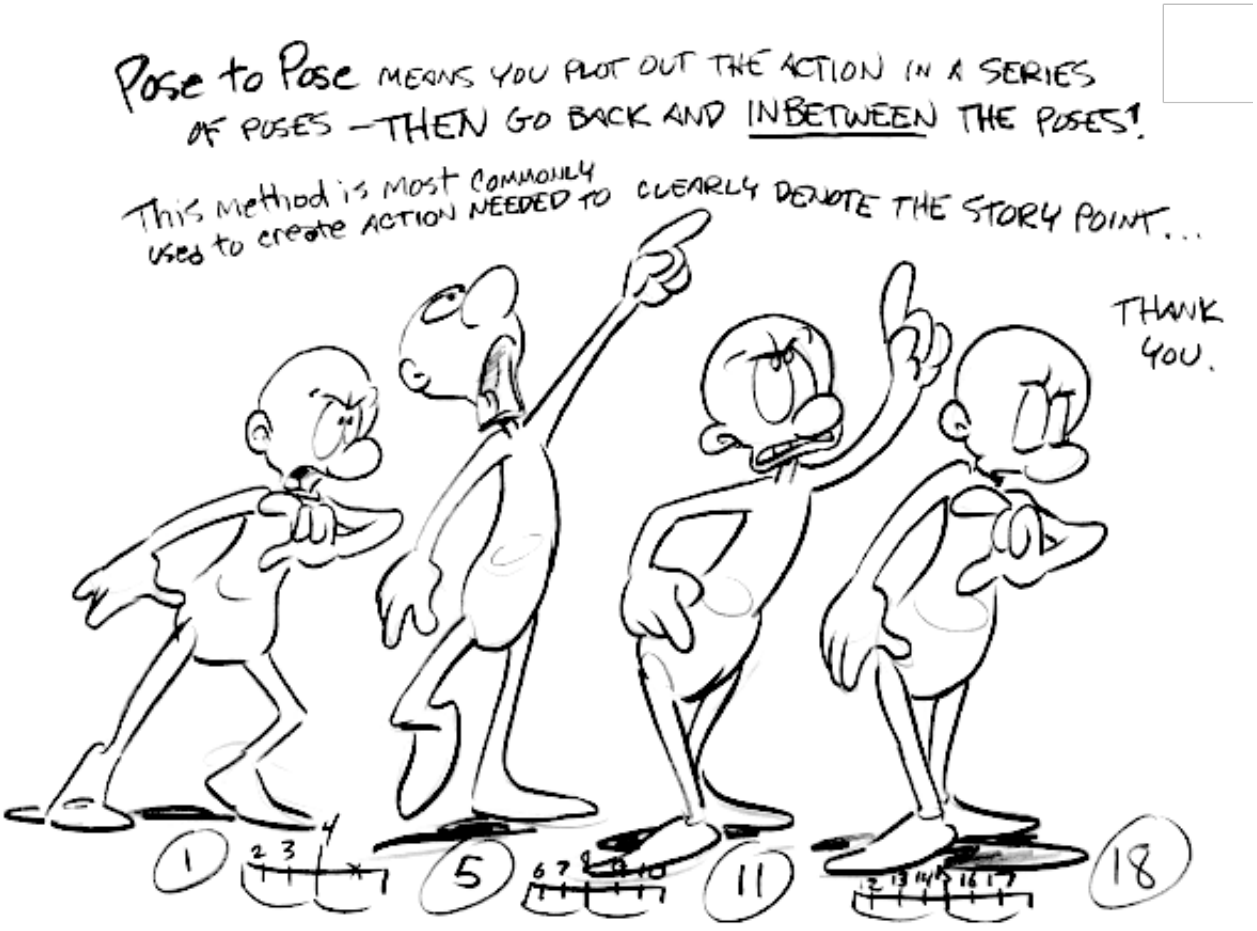

Figure 9: Pose to pose method

Interpolation

Similar to pose to pose, interpolation starts with the creation of key poses. The difference with the previous technique, nonetheless, is that once the key poses have been created, the interpolation between them is made

with tools that allow the artist to generate the transitioning frames. Evidently, this renders the process much more time-efficient, as only a few poses have to be created while the in-between frames are generated semi-automatically. On the other hand, it some cases it can allow for less management over the transitioning frames to the artist.

Motion capture

Motion capture, also known as mo-cap, is a technology used to capture and record real-time movements of people or objects. At its essence, it can be understood as a modern take on rotoscoping. Motion capture is widely used in various fields such as film, video games, sports, and medicine. The process consists in placing markers on the object or person that we want to use as a subject, and tracking their movements utilising cameras and sensors. The gathered data allows the creation of a 3D representation of the movement, which is then used to animate digital characters, among other uses in different fields. This method has been used to animated CGI characters in feature movies such as Avatar, or the Lord of the Rings.

Machine Learning Powered Animation

This section would not be complete without mentioning one of the most recent advances made in the animation field. The Metahuman Animator tool, developed by Epic Games, is transforming the animation process by leveraging machine learning. The AI-powered system has demonstrated the ability to generate lifelike and detailed facial animations in a matter of minutes, by simply using a video captured on a phone. This technology is based on Epic Game's MetaHuman modelling tool. The technology allows users to manually create highly detailed human models in Unreal Engine 5 [25]. Ultimately, the Metahuman Animator tool aims to facilitate the animation process, making it more efficient and accessible.

Figure 10: Unreal Engine's Metahuman Animator

7.2.3 The Post-production Stage

Finally, the post-production stage concludes the animated sequence. In conventional techniques, this phase includes editing the timing of the animation for smooth and natural movement, adding final sound effects and music, potentially incorporating any last-minute visual effects, as well as compositing. Compositing is the process of layering multiple images or videos to create a single scene. The animation is then rendered, converting it into its final video format.

8 State of the Art

This review aims to provide a comprehensive understanding of the current state of animation techniques, tools, and methodologies, with a particular focus on 2D animation. Our exploration will include both traditional hand-drawn techniques and the more modern computer-based approaches, which have broadened the possibilities for animators. Essentially, we will delve into the software and hardware tools essential for these processes, and provide an understanding of their features, capabilities and limitations.

8.1 Latest Studies

We will start with a literature review of the latest studies and advancements made in the context of two and three-dimensional animation. As it is expected, most of these newer techniques make use of Artificial Intelligence techniques to facilitate the process in one way or another. In fact, we have already come across some software tools that are also leveraging the potential of AI. While this section will solely focus on the latest studies, the tools themselves will be revisited in the following sections.

8.1.1 Automated animation & human-assisted advancements

While researching the newest papers related to animation, two categories became apparent. While some papers proposed tools that aimed to remove the human factor from the process as much as possible, others were presenting tools that enhanced the animator's capacities, by allowing them to work more efficiently.

In the realm of fully automated animation, there have been advancements both for 2D and 3D animation. For instance, in the case of three dimensions, the paper **"An Automated Generation from Video to 3D Character Animation using Artificial Intelligence and Pose Estimate"** [26] proposes a method that achieves to generate 3D characters from a video, in a completely automatic process. Besides Artificial intelligence, which is used for 3D model animation, they also make use of **pose estimation** techniques. Evidently, this makes the process of 3D animation much less time-consuming.

"AnimaChaotic: AI-based Automatic Conversion of Children's Stories to Animated 3D Videos" [27] is another recent paper exploring the power of AI, this time for animating children's storybooks. By using Natural Language Processing, the tool transforms scenes from the inputted books into 3D animated scenes. To do so, information is extracted from the book, processed, and placed on the scene before being animated using AI, among other techniques. The result is a fully animated 3D video that brings the story to life in a visually engaging way.

On the other hand, there are also advancements in tools that aim to enhance the animator's capacities. For instance, the paper **"SketchBetween: Video-to-Video Synthesis for Sprite Animation via Sketches"** [28] presents a model that learns to map between keyframes and sketched in-betweens to create fully rendered sprite animations. This approach aligns closely with the standard workflow of animation, allowing animators to work more efficiently by automating the labor-intensive task of creating in-between frames.

Another interesting development is the use of AI for style transfer, as discussed in the paper by Shan Li titled **"Application of artificial intelligence-based style transfer algorithm in animation special effects design"** [29]. This approach uses AI to automatically apply artistic styles to animations, which can significantly speed up the animation process and allow for a greater variety of artistic expressions. This not only enhances the animator's capacity but also opens up new avenues for creativity and innovation in animation design.

To conclude, the field of animation is seeing significant advancements in both fully automated and human-assisted tools. With the amount of available papers taking advantage of Artificial intelligence to make improvements in the field, it is clea with applications ranging from style transfer in animation special effects to automated generation of animations for healthcare purposes. At the same time, tools that enhance the animator's capacities are also being developed, with a focus on aligning with the animator's workflow and automating labor-intensive tasks. These advancements are broadening the possibilities for animators and have the potential to significantly impact the animation industry.

8.2 Software Tools for Animation

8.2.1 Blender

Blender[6] is a an open-source 3D computer graphics software toolkit. It allows users to go through the entirety

of the 3D pipeline. That is, it supports modeling, rigging, animation, simulation, rendering, compositing, and motion tracking, even video editing, and 2D animation. It is also a powerful tool for both 2D animation. It comes with numerous helpful features for animators. Among the main aspects of the program's animation functionality, we can include its character animation pose editor, Non Linear Animation (NLA) for independent movements, forward/inverse kinematics for fast poses, and sound synchronization. These features offer animators both flexibility and control, making the process much more efficient and precise.
Besides the animation functionalities, Blender's rigging tools also stand out. These tools include envelope, skeleton and automatic skinning, easy weight painting, mirror functionality, bone layers and colored groups for organization, and B-spline interpolated bones. These rigging tools are key for creating realistic and intricate animations, whether for characters or objects.

In addition, Python scripting in Blender allows for the creation of custom tools and workflows, automation of tasks, and more. For instance, Python scripting can allow the creation of complex geometric objects, procedural animation, or batch processing of large number of files. This can save a significant amount of time, especially for repetitive or complex tasks. It is also thanks to the scripting interface that comes with Blender that this project is possible in the first place.

Despite the aforementioned features, Blender has some limitations as well. One notable drawback is its inability to use multithreading for Python scripts, which can limit the performance of complex or computationally heavy scripts. This could potentially slow down a workflow, especially for larger and complex projects. Additionally, some users also may find Blender's interface intimidating, and its workflow can be less intuitive, especially at first. In other words, it has a steep learning curve, which can be challenging for beginners.

Despite these limitations, Blender is still a very popular choice among many animators due to its powerful features, open-source nature, and active community. Hence why it was the software chosen for the development of this thesis.

8.2.2 Autodesk Maya

Maya[] is another commonly used software application for animation. As opposed to Blender, it is mainly used for creating 3D animations, but supports modeling, simulation, rendering, and compositing as well. Furthermore, Maya offers a comprehensive set of tools that assist animators in creating both 2D and 3D animations. Another key difference with the previous software is the fact that Maya is not only a private software, it's also not free. The software belongs to Autodesk, a company with known for their multiple products for designers, engineers and architects.

In regards to animators, Maya's animation tools offer advanced features like a 3D visual animation editor, time editor, and motion path 3D markers for smooth animations. It also provides robust rigging tools, which are fundamental for character animation. Animators can create sophisticated, believable characters using the HumanIK rigging and skeleton system. Furthermore, the simulation and effects features include the Bifrost for procedural effects. Bifrost is a tool that makes it possible to build realistic effects in less time.

On the other hand, due to its high skilled toolset, Maya also has a steep learning curve. In fashion similar to Blender, it's the interface and functionality what can seem complex and intimidating for new users. Maya also demands high-performance hardware for smooth operation, especially for tasks involving high-resolution models, complex scenes, and advanced rendering.

Maya is widely used in the film and TV industry, and it's recognized for its advanced capabilities. Many of the best-known animated films and visual effects in live-action films were created using Maya, attesting to its powerful features and capabilities.

8.2.3 Adobe Animator

Adobe Character Animator is a desktop application software product from Adobe Systems. It's used to combine live motion-capture with a multi-track recording system to control layered 2D puppets drawn in Photoshop or Illustrator. It captures the user's facial expressions and movements in real time through a webcam and then applies these movements and expressions to the character. This allows the character to mimic the user's movements, creating a unique and intuitive animation process.

Adobe Character Animator assists animators by providing an intuitive, real-time animation system. You can turn any Photoshop or Illustrator character into an animated character using a webcam and microphone to

match your movements and voice to the character. This real-time rendering allows for a dynamic and interactive animation process. The software also offers various panels for different aspects of animation, such as Rig, Scene, Timeline, Controls, and Stream panels, each providing different functions for the animation process.

Adobe Character Animator is not open source, it is a proprietary software developed by Adobe Systems. It's also not free, it's part of the Adobe Creative Cloud, which requires a subscription.

The most notable quality of Adobe Character Animator is its live, real-time animation feature. It uses the animator's own expressions and movements to animate characters, making the animation process more intuitive and lively. It also has a unique feature where the animator's voice can be used to trigger associated animations, adding another layer of interaction to the animation process.

As for limitations, Adobe Character Animator might seem simpler and have fewer features than other animation software due to its unique approach to animation. While this simplicity can be a boon for beginners or those looking for a streamlined experience, it might limit the level of detail and control experienced animators can achieve.

8.2.4 Reallusion Cartoon Animator

Cartoon Animator[8] is a comprehensive 2D animation software that comes with a wide range of features and tools designed to assist both beginners and professional animators. The software is known for its unique ability to apply 3D motions to 2D characters, giving more depth and realism to traditional 2D animation. This feature is what makes Cartoon Animator one of the closest tools to what this thesis aims to produce.

Besides, Cartoon Animator offers a variety of other features to assist animators. Among them, the most relevant ones are motion capture, several software integrations, and ease of use. First, the motion capture allows users to use either a phone or a webcam for mapping their facial expressions into digital models. Secondly, the photoshop integration allows for quick conversion of image layers into animated two-dimensional objects. Nonetheless, the software is neither open-source or available for free, which limits its accessibility to some users although coming with a free trial version.

8.2.5 Cascadeur

Cascadeur[11] is a highly advanced physics-based animation software that incorporates AI algorithms to improve and streamline the animation production process. By using artificial intelligence, it not only creates more realistic animations but also helps animators increase their efficiency.

The software achieves this by employing its "motion analysis tools". These are advanced AI algorithms that are designed to analyze motion data meticulously, identifying key features such as joint angles and velocities. This detailed analysis allows for the creation of animations that are physically accurate and believable. Animators no longer have to guess or approximate these values, which can greatly reduce the time spent on this aspect of the animation process.

Cascadeur boasts several significant qualities that make it a noteworthy tool for animators. Chief among these is the software's ability to facilitate physics-based animation, which aids in the creation of high-quality, realistic animations. By automating some of the most laborious aspects of animation, such as motion data analysis and keyframe generation, Cascadeur significantly improves efficiency. Its motion analysis tools offer precise information on joint angles and velocities, enabling more lifelike movement rendering. Despite its sophisticated functionality, Cascadeur's interface is user-friendly, ensuring accessibility for animators and providing a plethora of tools and features to streamline the animation process.

However, it's important to acknowledge Cascadeur's potential limitations as well. Due to its high-end features, Cascadeur may pose a steep learning curve, especially for animators who are new to the realm of physics-based animation or AI tools. Its powerful nature also implies a need for robust hardware, potentially limiting its functionality for users with older or less capable machines. As proprietary software, Cascadeur's lack of an open-source model hinders user customization and community contributions to its development. While there is a free version of Cascadeur available, its functionality may be somewhat limited, which could prove restrictive for professional animators or larger studios.

8.2.6 Adobe Mixamo

Adobe Mixamo [9] is an innovative, cloud-based software service designed specifically for crafting and animat-

ing 3D characters. Its status as a free service, accessible with an Adobe ID, has cemented its position as an attractive option for many animators. However, as it's not an open-source software, customization potential and the ability to contribute to its development is somewhat limited.

There are several distinct benefits that Adobe Mixamo presents to animators. The software is highly user-friendly, making 3D character animation accessible even to those with limited prior experience in the field. A vast library of preset character models and animations acts as a significant timesaver, and these models are highly customizable to suit a range of creative needs. Furthermore, as a part of the Adobe suite, Mixamo can seamlessly integrate with other Adobe products like Adobe After Effects or Photoshop. This interoperability can enhance workflows and create a more cohesive creative process for users of these programs.

Nevertheless, potential limitations of Adobe Mixamo must also be considered. While the software does offer customization options, the scope of these may not meet the needs of projects that necessitate highly specific character designs or animations. As a cloud-based service, Mixamo is reliant on a reliable internet connection for access and operation, which could be problematic in areas with poor or unreliable internet service. Also, its status as proprietary software precludes community contributions to its development and restricts the degree of customization possible. Finally, Mixamo may not offer all the advanced features found in more comprehensive 3D modeling and animation tools, which might limit its applicability for more complex or professional-grade projects. Therefore, while Adobe Mixamo presents a highly accessible solution for 3D character animation, it's crucial to consider its potential limitations when deciding if it's the right tool for you.

8.3 Software Tools for Modelling

8.3.1 ZBrush

ZBrush[31] is primarily a digital sculpting tool, which utilizes a proprietary technology called "Pixol" to enable users to build their designs starting with a simple sphere or use customizable brushes to shape, texture, and paint virtual clay in a real-time environment. It stands out from other 3D tools due to its ability to handle models with an astronomical number of polygons without impacting performance, allowing artists to create incredibly detailed models.

ZBrush provides several unique features that make it a powerful tool for animators. Its sculpting workflow is incredibly intuitive and robust, allowing artists to sculpt intricate details with ease. The software's capacity to handle extremely high polygon counts enables the creation of incredibly detailed and realistic models. Furthermore, ZBrush offers excellent texturing and painting features, which further enhance the visual appeal of 3D models. Its compatibility with other 3D software also allows seamless integration into existing pipelines.

Despite its robust features, ZBrush does have some limitations. Its user interface is often seen as non-intuitive and can be quite overwhelming for beginners. Although it has extensive features for sculpting and painting, it lacks some of the animation and rigging tools present in more comprehensive 3D packages, requiring a separate tool for these tasks. It also has high system requirements due to the processing power needed to handle high polygon counts.

8.4 Software Tools for Pixel Art

8.4.1 Aseprite

Aseprite[32] is a proprietary, source-available image editor primarily designed for pixel art drawing and animation. It is available on multiple platforms including Windows, macOS, and Linux. The software offers different tools for image and animation editing such as layers, frames, tilemap support, command-line interface, and Lua scripting, among others. Aseprite's main design purpose is to create animated 2D pixel-art sprites, and it uses its own binary file type (.ase or .aseprite extensions) to store data. Its files can be parsed in various programming languages, and in game engines such as Unity and Godot. It also supports the export of images and animations to different file formats including PNG, GIF, FLC, FLI, JPEG, PCX, TGA, ICO, SVG, and bitmap (BMP)

Despite its versatile and powerful features, Aseprite is primarily focused on pixel art and may not be as effective for other forms of digital art. Furthermore, while it does support Lua scripting, this may require additional learning for users who are not already familiar with this programming language. It should also be noted that, while Aseprite has a wide range of export options, its native file format may not be directly compatible with all other software.

8.4.2 Piskel

Piskel[33] is a free, open-source pixel art editor that can be used directly in a web browser, although offline versions for Windows, Linux, and Mac OS X are also available. It is licensed under the Apache License. Piskel is designed to be beginner-friendly while still providing a robust set of tools for creating pixel art and animations, including features like onion skinning, live playback, and the ability to export to sprite sheets or gifs.

Piskel's browser-based design allows it to be used on any device with a modern web browser, making it highly accessible. It also provides a sleek interface that is easy on the eyes. The software includes an animation support feature, which allows for the definition of frame rates and the presentation of live playback. Despite its simplicity, Piskel has enough tools to be powerful and useful for both beginners and experienced users.

However, Piskel does have certain limitations. It lacks a true resizing tool, requiring users to resize the whole image or import and shrink an image from the export tool. There is no button for undoing actions; users must use the keyboard shortcut Ctrl+Z. Rotation is only made in predefined angles, and it does not support variable rotations or mouse-based rotation. It's also noted that Piskel does not work well with drawpads, and shading can be difficult to achieve.

8.4.3 Pixel Art rendering - Free blender addon

Finally, it is necessary we also mention another add-on for Blender that aims to do something similar to our tool, but exclusively for Pixel Art. Pixel Art rendering - Free Blender addon [35] is a Blender plugin that generates pixel art materials for 3D models. It allows the user to generate high-quality pixel Art from 3D models quite quickly. It is open source and very quite simple.

However, despite being quite simple, it requires some knowledge of Blender for making it work. Additionally, it currently only works with Blender's Eevee renderer. Since there are some online tutorials online, even beginners can understand how it works quite quickly.

Since this is one of the only add-ons we have found that is directly comparable to our tool, we will use it for comparison reasons later on.

8.5 Software Tools for Cel Shading

8.5.1 Cinema4D

Cinema4D [34] is a 3D modelling, animation, and rendering application developed by the German company Maxon. Notable features of Cinema4D include procedural and polygonal modelling, animating, lighting, texturing, rendering, and common features found in 3D modelling applications. It's known for its ease of use, quick rendering, and powerful, flexible options. Cinema4D is used in a variety of industries, including film, visual effects, science, architecture, and design.

This software's greatest strength is its wide array of applications. From creating complex 3D models and animations to rendering high-quality visuals, this software is capable of handling a broad range of 3D design tasks. It's known for its user-friendly interface and speed, making it an excellent choice for 3D artists at all skill levels. Furthermore, it offers robust integration with other popular software like Adobe After Effects and Adobe Illustrator, which expands its functionality and flexibility.

However, like any powerful software, it also has a steep learning curve, especially for beginners in 3D modelling. While it is easier to learn than some other 3D programs, new users will still need to spend considerable time learning to navigate and utilize its many features. The cost of the software might also be a barrier for some users, as it is professional-grade software with a professional-grade price tag.

8.6 Tools Summary

The following table shows a summary of the technologies explored, highlighting their main features.

Name	2D	3D	Open Source	Free	Main Pros	Main Cons
Blender	Yes	Yes	Yes	Yes	Supports full animation stack. Active community	Learning curve. Complex UI.
Maya	No	Yes	No	No	Advanced features for 3D animation. Widely used in professional industry	High system requirements. Learning curve.
Adobe Animator	Yes	No	No	No	Real-time animation Easy to use. Adobe Suite	Few features
Cartoon Animator	Yes	No	No	No	User-friendly. 3D-2D mapping. Motion capture.	Limited scope
Cascadeur	No	Yes	No	No	AI & physics-based animation. Task automation	Learning curve. High system requirements. Limited free version
Mixamo	No	Yes	No	Yes	User-friendly 3D models library animations library Adobe suite	Limited animations Online only.
ZBrush	No	Yes	No	No	Advanced features for 3D sculpting Widely used	Learning curve
Aseprite	Yes	No	No*	No	Designed for pixel art. Powerful tools for image and animation editing. Supports exports to multiple formats.	No custom brushes
Piskel	Yes	No	Yes	Yes	Works in any browser. Animation support. Simple for beginners. Open source.	Lacks a true resizing tool. No button for undo. Rotation is limited. Doesn't work well with drawing tablets.
Cinema 4D	No	Yes	No	No	Extensive features for 3D modelling, animating, lighting, texturing, and rendering. Used in the film industry	Learning curve

Table 8: Comparison of animation technologies

9 Proposed Pipeline

The previous sections painted an overall picture of the current state of the animation field. Now, before delving into the implementation details of the developed product, this section aims to clearly establish where the built tool falls exactly on the landscape.

We have seen a steady trend of improvements in the field thanks to the use of Artificial Intelligence, either by enhancing the animator's work or by substituting it almost completely. Additionally, the most used tools have been explored, including Blender, as that is the chosen software for the thesis.

The developed tool clearly falls in the category of tools that want to facilitate the work of animators by speeding up the most time-consuming parts of their work. As a result, they can focus on the artistic side of two-dimensional animation, with the benefit of it not taking several days of work to complete.

Therefore, in order to properly introduce the tool that comes with the thesis, this section will first explain the proposed workflow an animator would have to follow to obtain a 2D animation with satisfying visual quality, in a time-efficient way.

9.1 Model creation

This step demonstrates what a user who wanted to create a 2D animation from a 3D model made from scratch, would have to follow. However, it is worth noting that users who preferred to acquire a pre-made model from some source would be able to still use the product, and simply skip this step.

Essentially, the process of creating a three-dimensional (3D) model from scratch involves several important steps. Each of these steps contributes to the final outcome, which can range from still images to dynamic animations, which are our main focus.

Typically, the stages are modelling, rigging, animating, shading, compositing, and rendering. All these steps can be completed using software tools like Blender, which offers a wide range of tools and features for each phase. However, other applications such as **Maya** or **ZBrush** are also widely used.

9.1.1 Modeling

Modelling a mesh of polygons is the initial step in creating a digital representation of an object or character. Here we define the shape and form of the object in three dimensions using vertices, edges and faces in a 3D space. In the specific case of Blender, this can be achieved using the software's extensive set of modelling tools, such as extrusion, loop cut, and subdivision surface, among many others. Additionally, it comes with some basic shapes that allow for a quick start. Besides modelling, some artists prefer to use sculpting tools instead, in order to give shape to their models. Instead of working directly manipulating faces, digital sculpting comes with a variety of tools that help deform the geometry. This technique is done in a similar fashion to real-life sculpting with manageable materials such as clay, for instance.

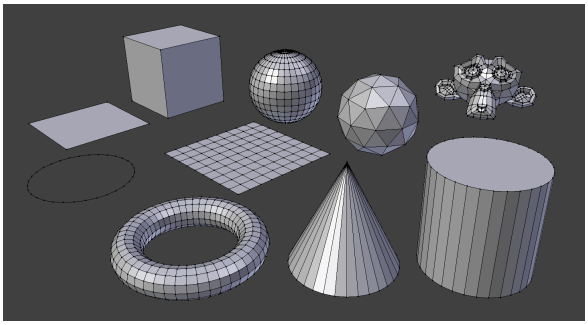

Figure 11: Blender's primitve meshes

9.1.2 Rigging

Once the 3D model is created, the next step is rigging. This process involves the creation of a digital skeleton, which can be manipulated to animate the model. In Blender, rigging is achieved by adding armatures to the model, which are essentially a series of interconnected objects, named "bones". These bones can then be attached to the model, in such a way that when manipulated, the bones, along with the model move imitating

the desired movement. For instance, in the case of animating humanoid characters, a skeleton that matched the character's anatomy would be necessary. Basically, it would need bones for the arms, legs, spine, and so on, which would be then attached to the corresponding parts of the 3D model. While manually rigging and animating a model can easily become very time-consuming, there are some alternatives available that facilitate the process to some extent. Adobe's Mixamo is the most known, and it is presented, among others in Section 8.2.

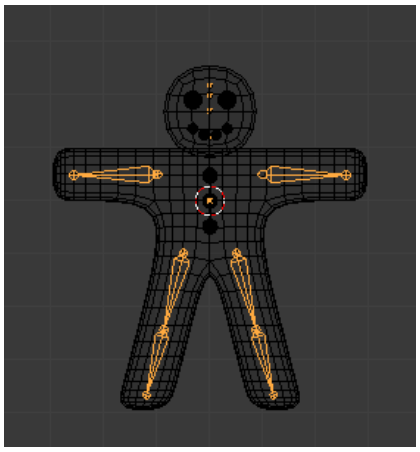

Figure 12: Rigged Blender Model - The bones are highlighted in orange

9.1.3 Animating

Of course, once the model has been rigged, it is time for actually animating it. Throughout this thesis, we have seen several ways an animator, or any other user could achieve this. They could either do it manually, for which the previous would be necessary, or use any of the many alternatives that allow for fast and automatic animation. That includes tools like Mixamo, the ones that leverage motion capture, such as DeepMotion. The AI-assisted ones like Cascadeur, or MetaHuman Animator could also be used here.

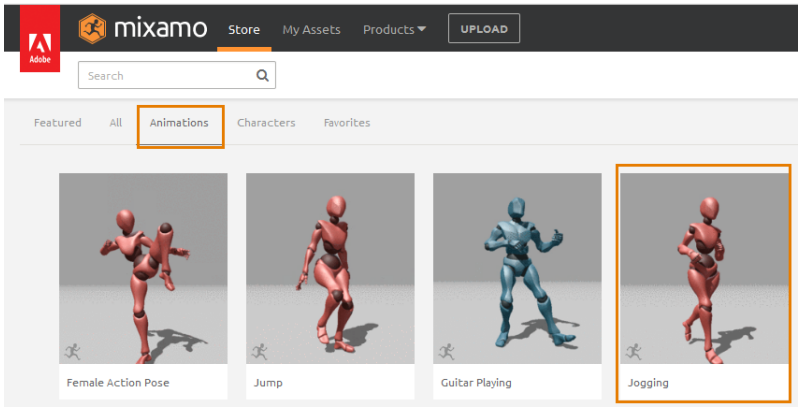

Figure 13: Some Mixamo animations

9.2 Plugin Usage

From this step to the end of the pipeline, the plugin assists the animator in achieving the final desired 2D animation. The add-on includes the operations for helping with colouring (or even recolouring) the model's parts, and then automatically applies the compositing configurations. Once everything is ready, the user can render the result In several formats.

9.2.1 Shading

Shading is the next phase, which involves defining the colour, texture, and material properties of the 3D model. When the final product is going to be a 3D model, this step adds depth and realism to it by simulating how light would interact with different types of materials. Blender offers a powerful shading system with a wide range of preset materials, such as glass, metal, and fabric, as well as the ability to create custom shaders on its Shader Nodes Editor.

Nonetheless in our case, the shading is going to be flat and the texturing minimal, since most of this detail would not be appreciated in the final 2D art styles. The plugin comes with colour selectors the artist can make use of to shade their model or scene if this has not been done previously. Additionally, there are other functions that are very useful at this step, especially for those users that have not made the model themselves. These will be discussed further in the implementation section.

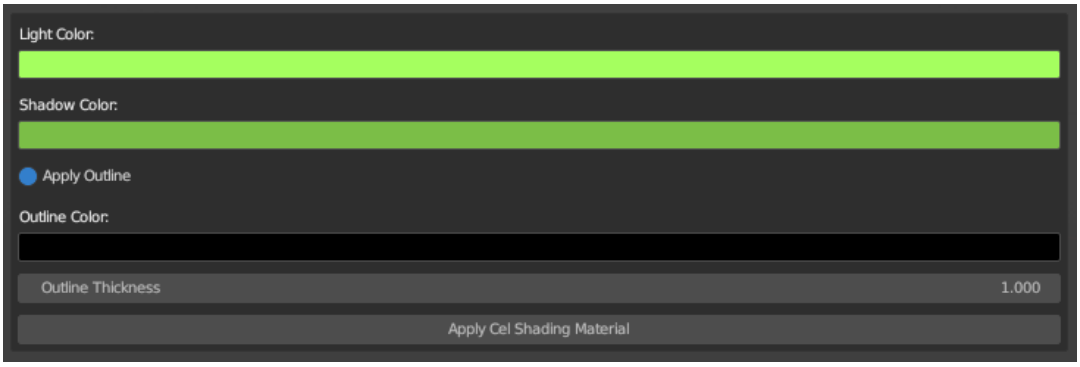

Figure 14: Shading section in the implemented add-on

9.2.2 Compositing

Compositing, which is the fifth stage, generally compliments what has been done during the shading phase. This step usually involves incorporating the 3D model into an environment and adjusting the colour, lighting, and other visual effects to create a cohesive and seamless image. In Blender, compositing is achieved using the compositor viewport; a post-processing tool that allows the adjustment of various aspects of the image such as colour grading, blurring, and adding effects like glare or depth of field.

With the add-on, this step becomes quite automatic, as the necessary compositing configurations for applying the desired art style, are implemented automatically as soon as the user renders the final scene.

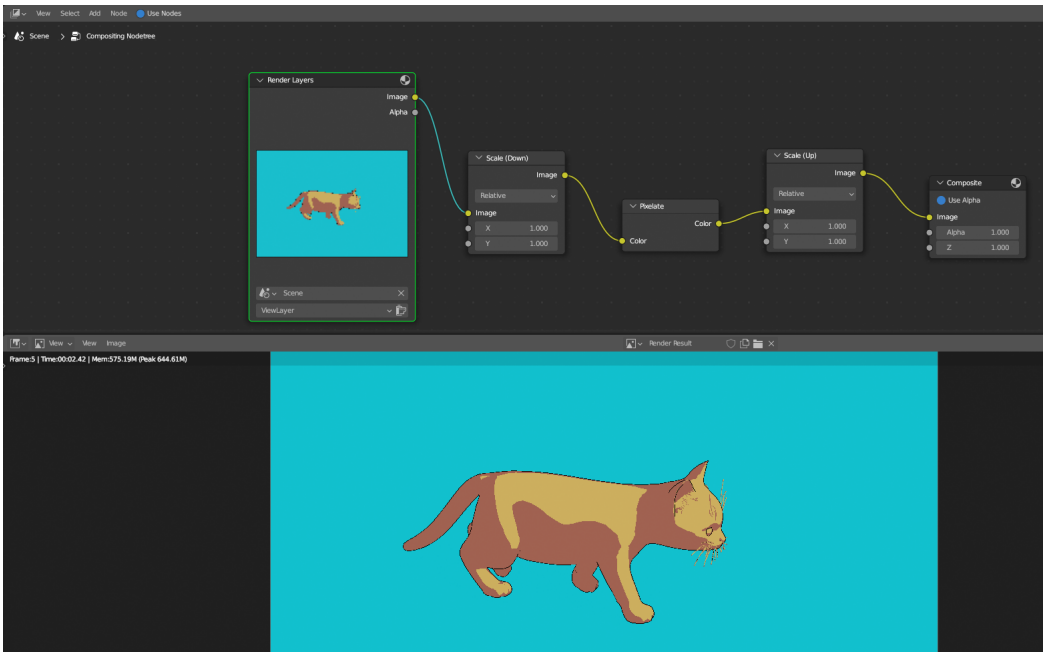

Figure 15: Resulting compositing nodes generated by the add-on

9.2.3 Rendering

Finally, the last step in the process is rendering. Rendering involves generating a two-dimensional image or animation from the 3D model and scene. The rendering process in Blender uses sophisticated algorithms to simulate the interaction of light with the objects in the scene, taking into account factors such as shadows, reflections, and textures. This can be a lengthy process when it comes to complex scenes, as the aforementioned

calculations could potentially take several hours to complete.

This final step is essentially equal when using the plug-in. The main difference is the addition of useful rendering formats the user can choose from, depending on which one adapts better to their project. There are three main formats. Firstly, a video playing the animation. Secondly, a directory with separate full-sized picture files, each one with an animation frame. And finally, the third format is a single image holding all the frames, displayed in a typical spritesheet format.

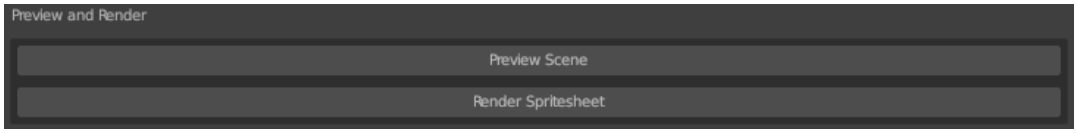

Figure 16: Preview and Rendering section in the implemented add-on

9.3 Pipeline Overview

All in all, the proposed pipeline makes use of the usually more manageable aspects of 3D animation to ease the 2D animation process. Admittedly, some parts of the pipeline, such as rigging and animating can still be very time-consuming and require high levels of expertise. However, we have seen some great techniques that automate specifically these stages, and that would not be supported in a fully two-dimensional workflow. By making use of the last advancements, all users, from beginners to professionals, can minimise the time spent on the most repetitive and long aspects of the animation process.

To summarize the whole pipeline, we have included a diagram in Figure 17 that includes all the stages, and separates them between the work that has to be done (or acquired) before the add-on usage, and what gets automated thanks to it.

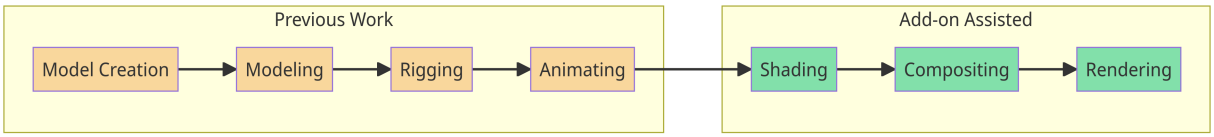

Figure 17: Full pipeline preview – from previous necessary work to stages automated by the add-on

10 Design and Implementation

This section delves into the core design and implementation of the created add-on for Blender. As stated, the add-on is developed with the main objective of transforming 3D animations into a variety of 2D styles. However, we will see that other features considered necessary ended up being implemented as well, making the final result a very useful tool for animators, as well as less skilled Blender users. In order to comprehend how each requirement of the add-on has been fulfilled, we will explore the function and purpose of each class and how they interplay with one another to generate the desired outcome.

10.1 Product Design

Before actually delving into the implementation details, it is important to discuss the Add-ons design and explain how this meets the requirements established early on. Therefore, this section will give a high-level overview of the project, highlight where the requirements are being met, and discuss other relevant design choices and UI/UX aspects.

10.1.1 Product Overview

The final product takes the form of an add-on developed for Blender, with the goal of automating and enhancing certain aspects of the animation process. This add-on has Python-based and it utilizes several Python libraries, as well as custom classes and modules. The list below includes all the Python libraries, as the self-programmed will be explained later on.

- **bpy**: The Blender Python API, a collection of modules that provide interfaces to Blender functionalities.
- **os**: A Python module that provides functions for interacting with the operating system.
- **math**: A Python module for mathematical operations.
- **numpy**: A library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.
- **webcolors**: A Python library for working with colour names and colour value formats defined by the HTML and CSS specifications for use in documents on the Web.
- **collections**: This Python module implements specialized container datatypes providing alternatives to Python's general-purpose built-in containers.
- **sklearn.cluster**: Part of the Scikit-learn library, provides the KMeans function for identifying clusters of data.
- **PIL (Python Imaging Library)**: Adds image processing capabilities to your Python interpreter.

Once the plugin has been added and executed, it shows as a panel within the Blender viewport. As per usual with Blender add-ons, the panel display quickly provides users with easy access to all functionalities. Essentially, its core feature lies in the ability to convert 3D animations into three different 2D art styles: pixel art, cel-shading, and outlined cel-shading.

In addition to these transformations, the add-on also comes packed with several auxiliary features aimed at facilitating the 3D-to-2D conversion process. These include mesh pre-processing, texture extraction, and texture simplification. Each of these features serves to prepare the 3D model for conversion and take a workflow that could easily take days to complete, completely finished in at most a few hours.

10.1.2 Product Requirements

During the planning phase, the requirements for this plugin were set, and here we strive to show how the final product meets these. Below, we revisit said list of final requirements with the intention of defining which features in the final product contribute to their satisfaction.

Number of explored art styles

The first requirement simply established that three art styles were going to be implemented, which is ultimately what took place. The final plugin currently supports Pixel Art, as well as Cel-shading with and without outlines. Additionally, thanks to the parametric features of the addon, these styles can be customised further.

Automating

As commented in the previous section, the plugin achieves to automate the equivalent of the frame drawing stage in a traditional 2D animation workflow. It supports everything from shading the model, to rendering it in several formats.

Multiple input cases

This requirement was added later on, as a need for it became apparent after discovering how many different sources a model could come from. Ultimately, automating would not be complete if the add-on didn't offer the same level of assistance and, more importantly, results to all of those. While the initial plugin assumed the user would have full control and knowledge over some of the model's aspects (texture, colour, vertex groups), the final version works in the same conditions with both custom and pre-made models, thanks to the mesh and texture processing features.

Efficiency

Also a quite self-explanatory requirement, efficiency was achieved in basically all the features the add-on incorporates. The slowest feature continues to be the one in charge of processing meshes, as it has to go over every single polygon, and multi-threading was not a viable option in Blender. Still, the process does not take more than a couple of hours to complete, and that is only for models with a large number of polygons.

Multiple output options

This requirement is achieved thanks to the three different output options the plugin provides. While two of them are already supported in Blender, the plugin adds a third one for users who plan to use the plugin for quickly making sprites.

Result Quality

Finally, the resulting quality is something that will be shown after the implementation details, where we prove how the results achieved with this plugin don't really differ from the ones that can be achieved with other tools or traditional techniques. More importantly, in the case of pixel art, the plugin successfully renders results that seem manually made, instead of a simple pixelated rendering of a 3D scene.

10.2 Product Implementation

Having understood the design and the requirements of the add-on, we now move forward to the details of the implementation. Ultimately, the goal of this subsection is to explain the technical details, the decisions made during the development process, and how each of the classes works to meet the stated requirements. The implementation phase of this project had a few challenges, each of which was tackled with a blend of Python programming best practices, careful utilization of Blender's API, and several third-party libraries that aided in various tasks.

The core functionality of transforming 3D animations into different 2D styles was implemented by creating custom operators and algorithms, which are encapsulated in their respective classes. Auxiliary functions like mesh pre-processing, texture extraction, and simplification were also programmed as distinct operators. Understanding the role of these classes and the flow of data between them is essential to comprehend the underlying mechanics of the add-on.

While **the full code is provided in the Annex**, some snippets of it will be displayed along with the implementation description, if deemed necessary for clarification reasons.

10.2.1 General structure

The add-on has been organised with a modular and intuitive directory structure. This layout promotes clean code and ensures that each module and class has a distinct responsibility. The structure, is as outlined below, and was done with implemented with maintainability and scalability in mind.

The entry point of the add-on is the **main.py** file. This file is responsible for defining the add-on's panel, registering and unregistering all operators, and importing necessary modules from the "Operators" directory.

The "Operators" directory contains several Python files, each representing a particular operator. These files include:

- **OP_mesh_process.py**
- **OP_style_celshading.py**
- **OP_style_pixelart.py**
- **OP_tex_extract.py**
- **OP_tex_extract.py**
- **OP_tex_simplify.py**

Each of these files contains operator classes that implement a specific functionality of the add-on, such as mesh processing, applying different 2D styles, extracting textures, and simplifying textures.

Within the "Operators" directory, there is a sub-directory named "Classes". This directory contains three Python files:

- **mesh_processor_split.py**
- **tex_simplifier.py**
- **style_pixelart_creator.py**

These files contain a class that encapsulates a specific functionality. Firstly, the file **mesh_processor_split.py** contains a class for splitting the mesh. Secondly, **tex_simplifier.py** includes a class for texture simplification. Finally, **style_pixelart_creator.py** houses a class for creating the pixel art style.

Below, Figure 18 displays the file structure just described, as a clear overview of the components and functionalities of the add-on.

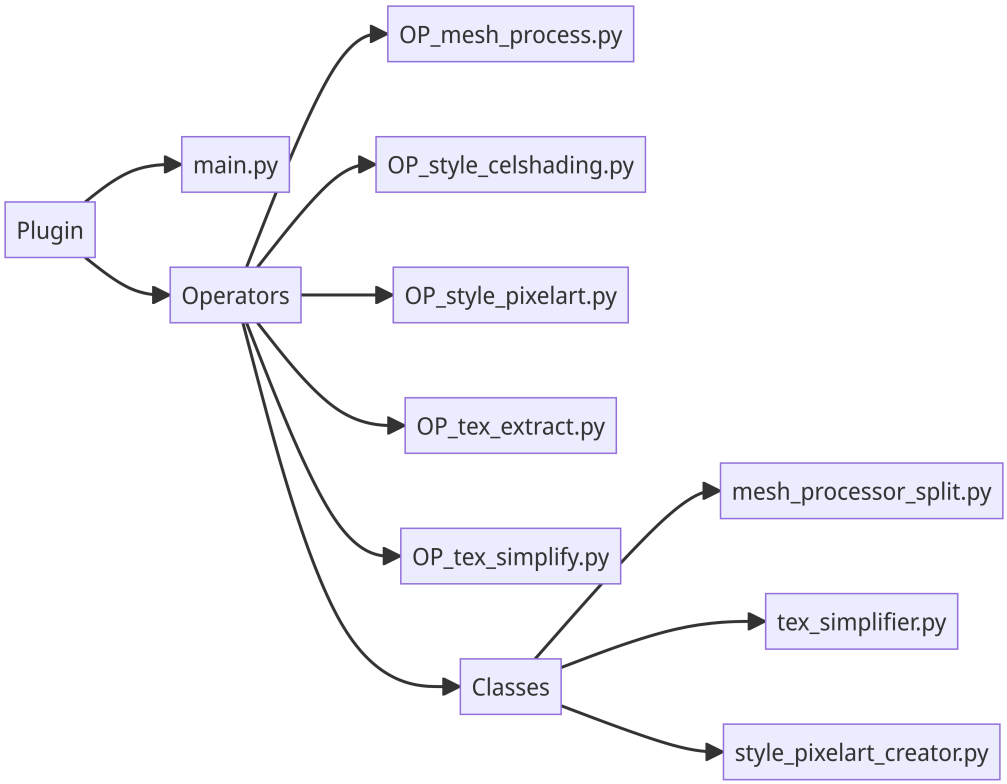

Figure 18: Add-on file structure

10.2.2 Main Panel

The **main.py** file functions as the entry point of the add-on, where all the other operator modules are imported and managed. This file includes information about the add-on in the **bl_info** dictionary, such as its name, author, version, Blender version compatibility, location within the Blender interface, description, warnings,

documentation URL, tracking URL, and category.

After this, all the operators from the **Operators** directory are imported. These operators provide functionalities for mesh processing, texture simplification, texture extraction, pixel art rendering, sprite sheet rendering, and cel shading.

The base function of this file is to host the **RENDER_PT_2DRenders** class, which is created as a subclass of **bpy.types.Panel**. It essentially represents the panel of this add-on in Blender's interface. This class contains methods for drawing the UI of the add-on. It sets up a panel in the 3D view's tool shelf, labelled "2D Renders". The **draw** method of the class defines the layout of the panel, displaying buttons and properties for all the imported operators and the features they implement. The images in Figure 19 below showcases the final design of the add-on in its current state.

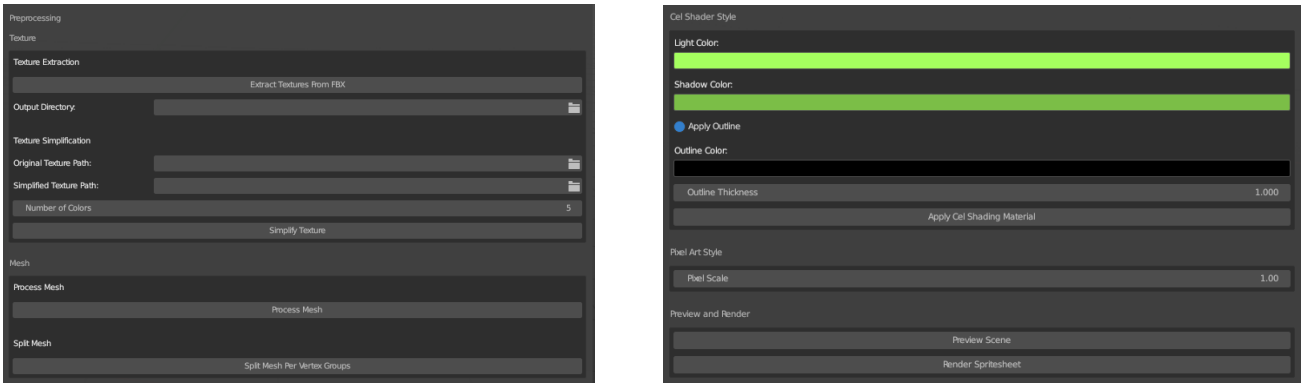

Figure 19: Add-on layout

The **register** and **unregister** functions are crucial within the Blender add-on ecosystem. They handle the initialization and cleanup of the add-on when it is enabled or disabled. Specifically, the **register** function is responsible for making Blender aware of the new functionalities the add-on introduces. It registers the panel and all the operator classes, and defines new properties for the scene. Conversely, the **unregister** function undoes these actions when the add-on is disabled, unregistering the panel and operators, and deleting the newly defined properties.

Lastly, the file checks if it is being run directly and if so, it calls the **register** function, effectively launching the add-on. This check is standard in Python scripts to prevent code from being run when the script is imported as a module.

10.2.3 Mesh Processing operators

Mesh Processing Operator Classes

The **OP_mesh_process.py** file contains two operator classes, **MESH_OT_process** and **SPLIT_OT_mesh**, which are used to manipulate 3D meshes within Blender.

MESH_OT_process is an operator for processing meshes. The **execute** method of the class is its main functionality, and it gets called when the operator is run in Blender. This method fetches the currently active object in the Blender context, creates an instance of the **MeshProcessor** class (from the **Classes** directory), passes the object and a texture path to it, and then calls the **process** method of the **MeshProcessor** instance.

SPLIT_OT_mesh is another operator class, which separates an original mesh into different parts based on their vertex groups. Similar to **MESH_OT_process**, this class also includes an **execute** method. In this case, however, the method retrieves the currently active object in Blender and creates an instance of the **SplitMesh** class instead. This class, along with the MeshProcessor, constitute the whole background of the mesh processing operators.

In addition, both classes have **register** and **unregister** methods, which register and unregister the operator classes, respectively. These are then called from the **main.py** file, as they are basic features for a Blender add-on to work properly.

MeshProcessor and SplitMesh Classes

The **MeshProcessor** class is responsible for processing a 3D mesh in Blender. The class contains various methods to aid in this processing.

The constructor of the class takes two parameters: the mesh object and the texture file. The class has static methods **get_pixel_from_uv** and **calculate_dominant_color**, which are used to get pixel values from a UV image and calculate the most common colour (dominant colour), respectively.

It also contains several instance methods. The **color_distance** method calculates the Euclidean distance between two colours, and the **find_closest_color** method uses this to find the colour in a given list that is closest to a given colour.

The **process** method is the main driver of the class. It loads the texture image, fetches the object's UV map and vertex group colours, and iterates over the polygons in the mesh object. For each polygon, it collects the colours of the pixels in the texture that corresponds to the polygon's vertices, calculates the dominant colour among them, and finds the vertex group colour closest to the dominant colour. It then assigns the polygon's vertices to the vertex group corresponding to this colour.

```
def process(self):

image = bpy.data.images.load(self.texture)
uv_map = self.obj.data.uv_layers.active.data

vg_colors = []
for vg in self.obj.vertex_groups:
    if vg.startswith('#'):
        rgb = tuple(int(vg.name[i:i+2], 16) for i in (1, 3, 5))
        vg_colors.append(rgb)
print(vg_colors)

for poly in self.obj.data.polygons:
    pixel_colors = []

    for loop_index in poly.loop_indices:
        loop = self.obj.data.loops[loop_index]
        uv_coord = uv_map[loop.index].uv
        pixel = self.get_pixel_from_uv(image, uv_coord)
        pixel_colors.append(pixel)

        dominant_color = self.calculate_dominant_color(pixel_colors)
        closest_color = self.find_closest_color(dominant_color, vg_colors)
        closest_color_hex = '#' + ''.join(f'{int(x):02x}' for x in closest_color)
        group = self.obj.vertex_groups.get(closest_color_hex) or self.obj.
            vertex_groups.new(name=closest_color_hex)
        vertices = [self.obj.data.loops[loop_index].vertex_index for loop_index in
            poly.loop_indices]
        group.add(vertices, 1.0, 'ADD')
```

Listing 1: OP$_{m}esh_{p}rocesssnippet$

It is worth noting that the **process** method is one of the less efficient in the whole code, given that the number of polygons in a model can increase really quickly. The slow performance mentioned earlier is essentially caused by this section. Although efforts to tackle this problem were made (limiting the calls to the **bpy** API, implementing batch processing solutions, etc), the function is still essentially sequential. And ultimately, until the Blender API isn't made thread-safe, the optimisation of this process and in consequence, the operator will be limited.

The **SplitMesh** class takes a mesh object and splits it into separate meshes according to their vertex groups. The **split** method of this class iterates over all vertex groups of the given mesh object. For each vertex group, it deselects all vertices, selects only the vertices in the group, duplicates them, and separates them into a new mesh. The operator that runs this class cannot be called unless the model's texture has been simplified first. The reason behind this is that these functions split the model into sections according to colour, which is extracted from the model's texture. By doing it this way, the resulting splitting follows a logical order.

10.2.4 Texture Extractor Operator Class

The class includes an **execute** method which is invoked when the operator is run. This method retrieves the active object in the Blender context and iterates over all materials of the object. If a material is node-based (that is, uses Shader nodes), it iterates over all nodes of the material. If it encounters a texture image node, it saves the image to the specified output directory.

```
def execute(self, context):
    obj = bpy.context.active_object

    for mat in obj.data.materials:
        if mat.use_nodes:
            for node in mat.node_tree.nodes:
                if isinstance(node, bpy.types.ShaderNodeTexImage):
                    img = node.image
                    if img is not None:
                        output_file = os.path.join(self.output_directory, img.name +
                            '.png')
                        img.save_render(filepath=output_file)
    return {'FINISHED'}
```

Listing 2: Texture extraction snippet

10.2.5 Texture Simplifier Operator

The **TextureSimplifier** class is designed to simplify a texture image by reducing its colour palette. This is typically done to create a more stylized or simplified version of a texture or to reduce the complexity of a texture for computational reasons. This class has the following properties:

- **image_path**: The path to the texture image which is to be simplified.
- **destiny_path**: The destination path where the simplified image will be saved.
- **n_colors**: The number of colours to use in the simplified image. This parameter controls the complexity of the colour palette of the simplified image.

Additionally, the class includes the **quantize_image** method, in charge of performing the colour quantization of the image. It uses the k-means clustering algorithm to find the most dominant colours in the image. The image is then recreated using only these dominant colours, effectively reducing the colour palette of the image. The quantized image is saved to the specified destination path and returned by the method, along with the dominant colours used. This is where the KMeans module enters into action, as it provides us with a very quick and clean way of achieving the simplification

```
def quantize_image(self):
        """
            Quantizes the colours in the image using K-means clustering.

            Returns:
                quantized_image (PIL.Image.Image): The quantized image.
                dominant_colors (numpy.ndarray): The dominant colours found by K-
                    means clustering.
        """
        image = Image.open(self.image_path)
        image = image.convert("RGB")

        pixel_data = np.array(image)
        pixel_data = pixel_data.reshape(-1, 3)

        kmeans = KMeans(n_clusters=self.n_colors)
        kmeans.fit(pixel_data)

        quantized_pixel_data = kmeans.cluster_centers_[kmeans.labels_].astype(int)

        quantized_pixel_data = quantized_pixel_data.reshape(image.size[1], image.size
            [0], 3)

```

```
        quantized_image = Image.fromarray(np.uint8(quantized_pixel_data))

        quantized_image.save(self.destiny_path + "__quantized.png")

        return quantized_image, kmeans.cluster_centers_
```

Listing 3: Texture simplification snippet

10.2.6 Pixel Art Operator

Preview Operator

The **RENDER_OT_PixelArtRender** class extends the **Operator** class in Blender's Python API and provides a custom operator for rendering pixel art in Blender. The **execute** method, which is called when the operator is run, calls a function from the **PixelArtNodeCreator** class to set up the nodes for pixel art rendering and then invokes the default Blender render operator.

Scene Renderer

This class is another operator for creating pixel art. This time, however, it is focused on creating a sprite sheet of the animation frames.

The **execute** method sets up the node tree for pixel art rendering using **PixelArtNodeCreator**, then calculates the dimensions of the sprite-sheet image based on the aspect ratio of the scene, the number of frames in the animation, and a specified maximum number of frames per row.

It renders each frame individually and then pastes them into the correct position in the sprite-sheet image. The result is a PNG image file where each frame of the animation is arranged in a grid, which is saved to a specified export path.

Compositing Nodes Creator class

The **PixelArtNodeCreator** class is used to set up the compositing node tree for pixel art rendering in Blender. The class contains a single static method, **create_pixel_art_nodes**, which configures the node tree for a given Blender context. This method scales down the render, pixelates it, scales it back up and then composites the result, effectively creating a pixel art effect. The degree of pixelation can be controlled via the **scale_value** attribute of the scene. The code snippet, along with the image in Figure 20 shows the implementation as well as the resulting nodes added.
pixel_art_nodes snippet – downscaling,

```
scale_node_1 = nodes.new(type="CompositorNodeScale")
scale_node_1.label = "Scale (Down)"
scale_node_1.space = 'RELATIVE'
scale_node_1.inputs[1].default_value = 1 / scale_value
scale_node_1.inputs[2].default_value = 1 / scale_value

pixelate_node = nodes.new(type="CompositorNodePixelate")
pixelate_node.label = "Pixelate"

scale_node_2 = nodes.new(type="CompositorNodeScale")
scale_node_2.label = "Scale (Up)"
scale_node_2.space = 'RELATIVE'
scale_node_2.inputs[1].default_value = scale_value
scale_node_2.inputs[2].default_value = scale_value
```

10.2.7 Cel Shading Operator

This class is a custom operator for creating and applying a Cel Shading material to the currently selected object in Blender. It extends the **bpy.types.Operator** class.

In the **execute** method, it retrieves the relevant properties from the scene, such as light color, shadow color,

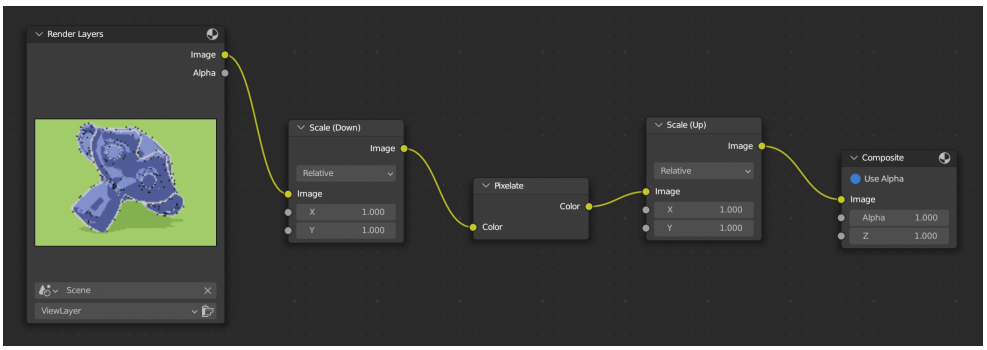

Figure 20: Resulting Compositor nodes

whether an outline should be applied, the position of the shadow, outline thickness, and outline color.

It then creates a new material and configures its node tree to create a cel-shading effect. This is achieved by using a **ShaderNodeBsdfDiffuse**, **ShaderNodeShaderToRGB** and a **ShaderNodeValToRGB** node, the last of which is configured with constant interpolation to achieve the hard colour transitions characteristic of cel-shading. The code responsible, as well as the resulting nodes are displayed below in figure 21.

```
# Create new nodes for cel shading material
diffuse_node = nodes.new(type="ShaderNodeBsdfDiffuse")
shader_to_rgb_node = nodes.new(type="ShaderNodeShaderToRGB")
color_ramp_node = nodes.new(type="ShaderNodeValToRGB")
color_ramp_node.color_ramp.interpolation = "CONSTANT"
color_ramp_node.color_ramp.elements[1].color = light_color
color_ramp_node.color_ramp.elements[0].color = shadow_color
color_ramp_node.color_ramp.elements[1].position = shadow_position
material_output_node = nodes.new(type="ShaderNodeOutputMaterial")
```

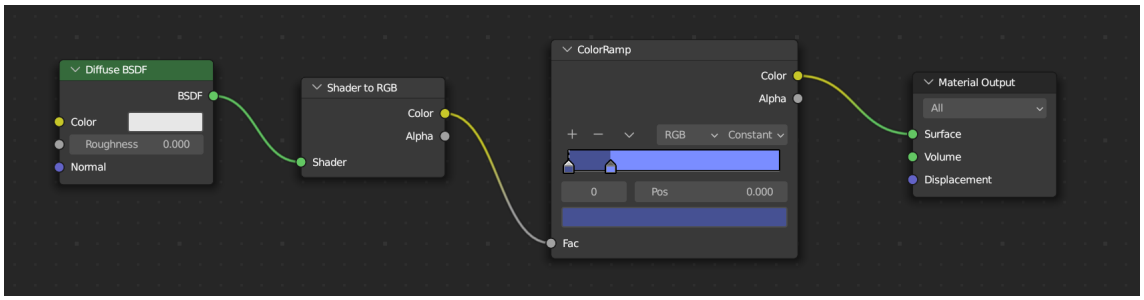

Figure 21: Resulting Shader nodes

If the **apply_outline** property is set to true, an outline is also added to the object. This is achieved by creating a new material with an Emission shader and applying it to a duplicate of the object geometry that is slightly scaled up and only rendered where it does not overlap the original object (backface culling).

As usual, **register** and **unregister** methods are used for registering and unregistering the operator with Blender's API.

11 Results

In the following section, we present the outcomes obtained with the elaborated add-on we just described. Our focus will be on evaluating the quality and authenticity of the pixel art, and cel-shaded graphics that our tool generates, comparing them with similar outputs produced by other tools and methods. We aim to provide an in-depth analysis that not only emphasizes the visual aspects but also considers the efficiency of the plugin. Key performance metrics such as execution times will be monitored and reported to compare them to the other observed tools.

Admittedly, it can be challenging to strive for objectivity when we are ultimately comparing artistic visual results. As a consequence, our comparison will try to go past a simple visual comparison and explore other metrics that will be defined shortly. In all, this evaluation is designed to paint a complete picture of the tool's capabilities, strengths, and potential areas for improvement.

11.1 Comparison Metrics

As mentioned, this comparison aims to go beyond the visuals. Therefore before we start we will quickly define the areas of comparison we will explore when comparing the tool to others.

- **Visual Quality Comparison:** Compare the quality of the assets generated by the implemented plugin to those created by other tools. This will have to be done in a segmented way since as we have learned from the literature review, no tools that implemented more than one art style in a similar fashion to ours, were found. So we will organise this comparison by the art style.
- **Performance:** Measure the execution times for the compared tools. Specifically, we will focus on evaluating how the complexity of the input (by number of polygons, and frame length) impacts the time required to generate the assets.
- **Ease of Use:** Consider the user interface and experience of your plugin in comparison to other tools.
- **Versatility:** Evaluate the range of different styles and looks the plugin can achieve, and compare this versatility to other tools. The ability to generate diverse assets can be a significant advantage since it provides more possibilities to the users in a single tool.

11.2 Tool Comparison

Unfortunately, during the state-of-the-art review, we did not find any tools we could fully compare to the one we implemented as they usually focus on a single art style. Nonetheless, we did find some interesting tools that can be used to reach at least some of the metrics we aim to study.

Overall, the following list displays all the technologies that we will use to compare at least some aspects of the plugin we implemented.

- Cinema4D
- Pixel Art Render Addon

11.2.1 Visual Comparison

We will start our comparison with the most straightforward of the measures. Given that all the tools are applicable here, we will structure them by art style and display the results obtained. We will use a simple pre-made model of a cat for these comparisons.

Pixel Art

The two following images represent a single frame of an animated cat walking. All the controllable parameters have been set equal between the two models. Therefore the only differences in the results are due to the add-on's implementation itself, which is what we aim to compare.

As seen in the figures, the differences between the two results are relatively minimal. On the left, the rendered frame has a less detailed shadow, and the transition between colours is sharper. The opposite happens with the frame on the right where a gradient is achieved thanks to the combination of the lighter and darker colors. Ultimately, they both are pixel art-styled cats of rather similar quality.

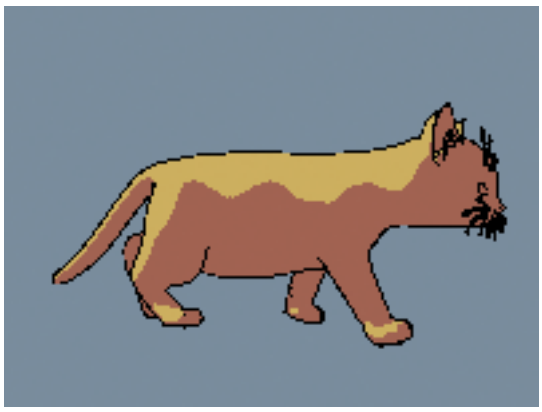

Figure 22: Our add-on

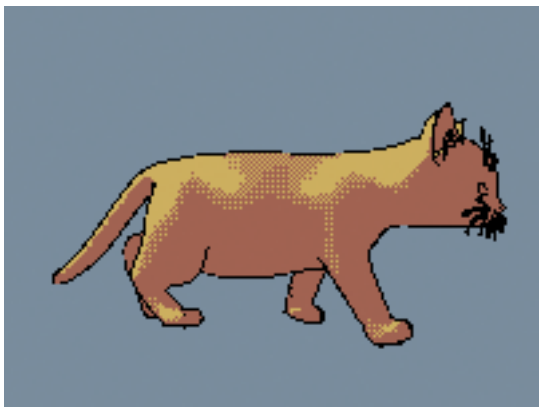

Figure 23: Pixel Art Render add-on

Cel Shading

Similarly, the three images below show the differences between the results obtained between our Blender add-on **(Figures 24 and 25)** and the result obtained with Cinema4D (**Figure 26**). It is worth noting that since this time the software used to build and render the scene was different, there might be some discrepancies in the lighting, shading and positioning of the cups used as an example. Besides these irrelevant discrepancies, the quality of the results is once again similar.

Figure 24: Non outlined

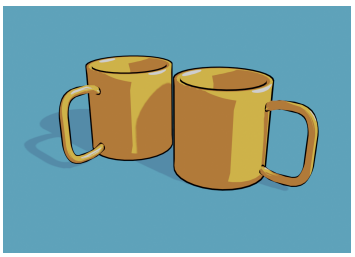

Figure 25: Outlined add-on

Figure 26: Cinema4D render

11.2.2 Performance metrics

Considering the background this thesis has, performance is among the most relevant metrics to take into account. Especially because low performance could be detrimental to the overall quality of a tool. For that reason, we will take a look at the execution times for all the methods included in the developed plug-in, and then compare those to the execution times obtained with other software. Evidently, we will not be able to make a one-to-one comparison, since the processing features are not applicable to the alternatives. However, we will still pay special attention to those methods as they are the ones that take longer to execute and scale quicker.

To obtain accurate measurements of execution times for the functionalities within our plugin, we will make use of Python's built-in time module. More specifically, we will utilize the **time.perf_counter()** function. This function offers an accurate counter for timing the duration of small code snippets. Basically, the function will allow us to encapsulate the exact parts of our plugin whose execution times we are interested in measuring. We will record the time immediately before and after the execution of the code snippets. The difference between these two-time points will give us the total execution time. This methodology will also be used when testing the alternative pixel art add-on for comparison purposes, given that we have access to the code.

To properly study this aspect of the implemented code, it is important to distinguish the operators that will remain constant no matter the input model, from those that do not. On one hand, classes like the one that generates the panel on the viewport, or those that alter some general property on a model, such as its material, are expected to have a practically linear execution time for models of all sizes. On the other hand, however, we expect a linear increase for processing operators. This is because those methods iterate over entire elements, namely textures, 3D models and vertex groups.

Texture Extraction

First, we have examined the impact of the number of materials on texture extraction time. From Table 9, it is clear that the texture extraction time increases almost proportionally with the number of materials, just as could see in the previous method.

Materials	Texture Extraction Time (s)
1	3.71242
2	7.52235
3	11.00453
4	14.62415
5	18.79023
6	22.01098
7	26.11375
8	29.24046
9	33.82847
10	36.82219
11	40.52239
12	44.83291

Table 9: Estimated Texture Extraction Times for Various Numbers of Materials

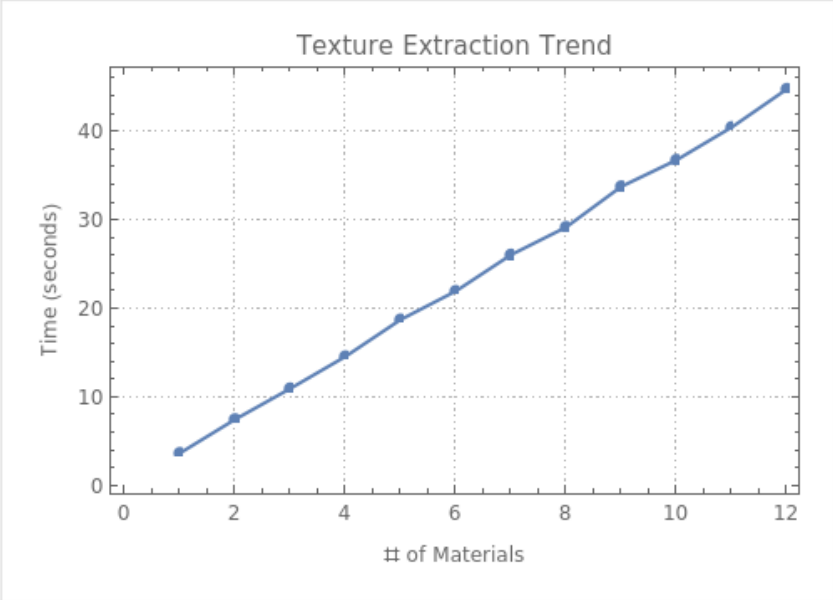

Figure 27: Texture extraction Trend

Texture simplification

Secondly, we explore the effect of image resolution on texture simplification time. Table 10 and Figure **??** confirm a less linear association between the two factors. As we know, this operator makes use of methods from the Python Kmeans module in order to successfully group the colours in a texture by their closest colour on the array. It's these extra levels of complexity that explain the resulting figures.

Image Resolution	Texture Simplification Time (seconds)
256x256	0.0638597111
512x512	0.2535514944
1024x1024	1.0106425664
2048x2048	4.0585248372
2560x2560	6.3452354431
3072x3072	9.1278542892
3584x3584	12.4090023754
4096x4096	32.4248766659
4608x4608	45.6206213013
5120x5120	61.5463650711
5632x5632	80.1478479067
6144x6144	101.440300987

Table 10: Texture Simplification Times for Various Image Resolutions

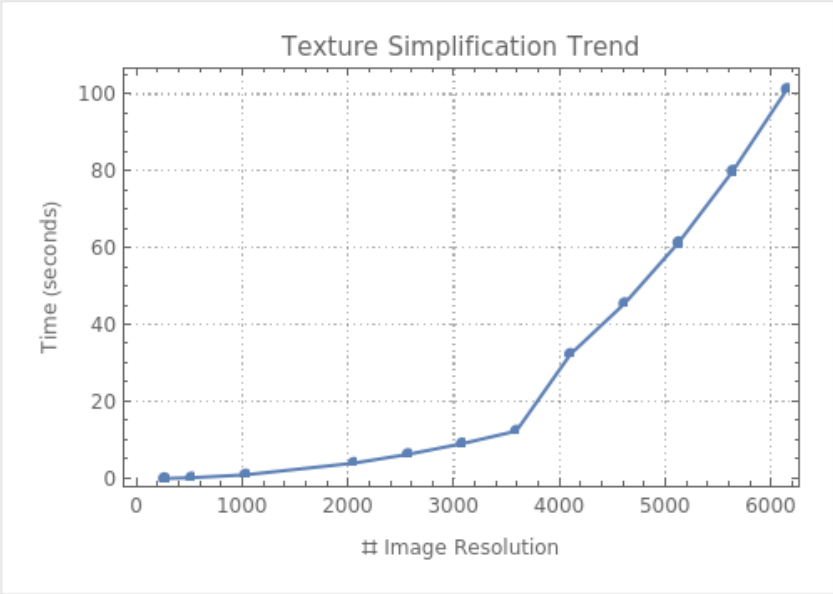

Figure 28: Texture Simplification trend

Mesh Processing

Next, we present a study on the relationship between the number of polygons and the time required for mesh processing. Table 11 depicts how the mesh processing time scales almost linearly with the number of polygons. This is pretty much what was expected, given that the executed code effectuates the operations linearly as almost no parallelization is possible. This linear relationship can be seen more clearly in Figure 29.

Polygons	Mesh Processing (s)
5	4.15668
50	38.71295
500	378.24887
1000	758.03425
2000	1601.0685
3000	2282.10275
4000	3072.137
5000	3856.22277
7500	5685.25688
10000	7495.3425
12500	9475.42813
15000	11370.51375

Table 11: Execution time comparison of operators

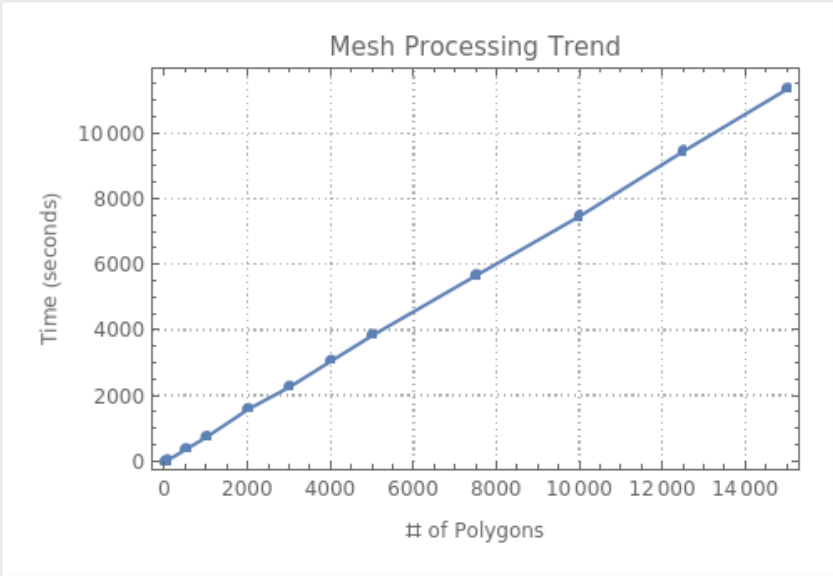

Figure 29: Mesh Processing trend

Mesh Splitting

To finish, here we analyze the relationship between the number of vertex groups and the time taken for mesh splitting. As demonstrated in Table 12, we see another linear relationship takes place in this case, as the executed code simply iterates over the array of vertex groups.

Vertex Groups	Mesh Splitting Time (s)
1	0.0270952107
2	0.0512288725
3	0.0744417318
4	0.1020425106
5	0.1282031525
6	0.1473553715
7	0.1804061933
8	0.2045527341
9	0.2321535129
10	0.2572891747
11	0.2814317355
12	0.3065762972

Table 12: Estimated Mesh Splitting Times for Various Numbers of Vertex Groups

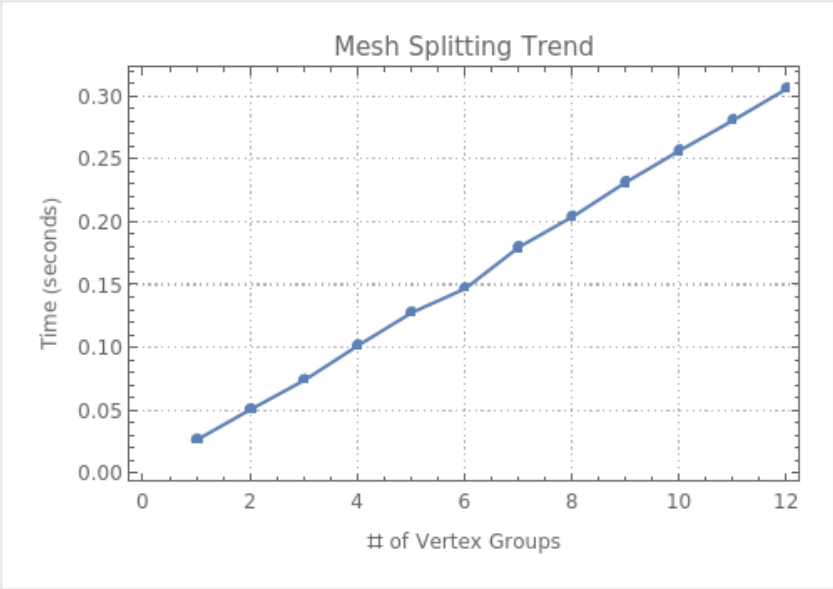

Figure 30: Mesh Splitting trend

Art Styles

The operators used for creating and applying the styles to meshes on the scene cannot be scaled as the number of nodes that need to be created is always fixed. As a result, instead of measuring a trend, several measurements were taken for the three art styles, and we obtained the following results. Needless to say, these functions are extremely quick to execute.

- **Pixel Art:** 3.62 milliseconds
- **Cel Shader (no outline):** 0.95 milliseconds
- **Cel Shader (outline):** 1.68 milliseconds

Additionally, the following list contains the execution times for the pixel art add-on and snippets we are comparing or tool to. The differences here are not very significant for the cinema4D results. However, it is slightly more notable when it comes to the pixel art add-on. This is most definitely due to the fact that the add-on does create a few more nodes when creating and assigning the material. Once again, these measurements were taken several times, and what we gather here is an average.

- **Pixel Art add-on:** 7.22 milliseconds
- **Cinema4D Cel Shader:** 0.91 milliseconds

Having observed all the relevant measurements, we can confirm what was expected already; Our model performance scales up linearly. For all the execution times we took, the correlation between the execution time and the scaling factor was almost perfectly linear, which finds an explanation for the simplicity of the code behind it. While a linear time complexity is not ideal, once again we find ourselves limited by what's supported in Blender and what is not. However, except for the Mesh process trend, which can potentially get to the largest number of polygons displayed in the graph, the other operators still perform in reasonable amounts of time for large scaling factors. This is however a very important aspect of the add-on and will be revisited eventually if any new and better implementations arised.

11.2.3 Ease of Use

In a similar fashion to Visual comparison, we find that comparing the ease of use for tools is in a way subjective. This is due to the fact that ease of use is a multifaceted factor that goes beyond just the simplicity of the tool's operations. It encloses a range of elements including the tool's accessibility, intuitiveness, learning curve, and the availability of support resources such as documentation and tutorials. For instance, a tool with a user-friendly interface that clearly labels and organizes its features can significantly enhance its ease of use, as users can navigate through the tool with less effort and confusion.

In comparing our add-on with other tools, we will look into these aspects. We will evaluate the design and

organization of the tool interfaces, the availability and quality of documentation and tutorials, as well as the complexity or simplicity of the tasks required to generate the desired outputs.

It is also worth noting that this comparison is much more logical between two add-ons than between an add-on and a completely dedicated software such as Cinema4D. Thus, when comparing our tool with Cinema4D we will keep it quite general, and focus on those elements that are applicable to both.

Accessibility

Between the Add-ons:
Both our developed add-on and the alternative are quite accessible in terms of their availability. Being open-source, they are freely available for users to download, modify and use as they wish. They both also come with commented code which helps in understanding the underlying processes and thus, facilitates any potential modifications or updates that users might want to make. Furthermore, they each have accompanying documentation or tutorials, which guide users on how to effectively use the tools.

Between the Add-on and Cinema4D:
Comparing the accessibility between our add-on and Cinema4D, our add-on clearly has an edge as it is both free and open-source. The users can freely use, modify, and learn from it. Cinema4D, on the other hand, is proprietary software, not open-source, and users need to pay for it after a 14-day trial period. While the cost of Cinema4D can be justified by its extensive feature set, for users who only need the functionality provided by our add-on, Cinema4D may be a more expensive and less accessible option.

Intuitiveness

Between the Add-ons:
When it comes to intuitiveness and user-friendly operations, our add-on stands out given that it only requires a few button presses from the user to achieve the desired results, making it more beginner-friendly. On the other hand, the alternative add-on, while also featuring a simple interface, requires more user input and consequently, a deeper understanding of the processes behind it. This increases the complexity for users who are new to Blender, although admittedly, it also provides more control to those more experienced.

Between the Add-on and Cinema4D:
Our add-on proves to be more intuitive in comparison to Cinema4D. The cinema4D environment is similar to Blender and therefore can be quite intimidating to new users. The vast array of tools and options could potentially confuse users, especially beginners. However, it's important to note that this comparison may not be entirely reasonable as one is a simple add-on focusing on a specific set of functions while the other is a complete 3D modelling, animation, and rendering software with far more extensive capabilities.

Learning Curve

Between the Add-ons:
In terms of the learning curve, both add-ons are fairly low, making them easily adaptable by users. However, the context of their usage is still within Blender, and as we have discussed it brings its own challenges. While these add-ons may simplify a small part of those challenges, navigating and utilizing the rest of Blender's features is still a struggle for beginners.

Between the Add-on and Cinema4D:
Regarding the learning curve, our add-on is once again, evidently lower. This is due to its specific purpose and simpler operations in contrast to Cinema4D's more diverse and advanced functions. Cinema4D, being a complete 3D software package, requires a significant investment of time and effort to master, much like Blender.

11.2.4 Versatility

An important aspect to take into account when comparing tools is their versatility. This refers to the range of different tasks or operations that a tool can perform, and how adaptable it is to various user needs. In this regard, our add-on displays a significant advantage over the alternative add-on, as it provides both pixel art and cel shading capabilities. This feature allows users to switch between different styles, depending on their

project requirements, without needing to resort to different tools. This adds to its appeal, particularly for users looking for diverse styles in a single package.

The alternative add-on, in contrast, focuses exclusively on pixel art. While this can be seen as a limitation, it also implies a concentrated effort on refining this particular feature. Users interested in pixel art and wanting to delve deeper into its intricacies may find this level of specialization appealing. However, the lack of cel shading capabilities can pose limitations for those requiring a wider array of stylistic options within the same tool.

On the other hand, the versatility of Cinema4D is beyond question. It is a comprehensive 3D software, with capabilities extending far beyond pixel art and cel shading. In terms of versatility, Cinema4D clearly outshines both add-ons, including ours. However, this broad scope can be both an advantage and a drawback, depending on the user's requirements. For individuals seeking a focused tool for pixel art and cel shading, our add-on could be more appropriate, whereas those requiring a broader range of functionalities would find Cinema4D more suitable.

Overall, the versatility of a tool does not only depend on the number of features it possesses but also on how well it caters to the specific needs of its users. In this context, while Cinema4D is a highly versatile software, our add-on holds its own as a specialized, user-friendly tool for creating pixel art and cel shading in Blender.

11.3 General Results

This section will be dedicated to displaying general products that have been obtained with the plugin we implemented. The aim here is not to compare. Instead, the importance of this section lies in showcasing more interesting outcomes generated using our add-on. Not only will it provide concrete, visual evidence of what our tool can achieve, but it also allows us to demonstrate the range of its capabilities.

This way, we also allow the reader to take a look at those features that were purposefully not included in the previous section due to the lack of counterparts to compare them with. This basically includes images of the results of processing meshes, simplifying textures, and exporting the results as sprite sheets, ready to use in video games.

In order to display these results in a coherent way, we will sequentially apply the features of the add-on to the same model, and see the partial, as well as final results. Figure 31 showcases the original model as it was acquired from Mixamo.

Figure 31: Mixamo model used for partial results demonstration

11.4 Texture Simplifying

After extracting the texture from the model we acquired from Mixamo, we proceed to simplify the texture and prepare it for processing. The following images display the original texture as well as the resulting one, after simplification of the colours into a number the user can pre-select. In our case, we chose to have 6 different colours.

We can see how in the image on the right, all the extra details disappear from the texture, giving it a more flat look. This facilitates the process of splitting the mesh by colours greatly since the differences between them are now much more clear.

Figure 32: Original Texture

Figure 33: Simplified Texture

11.4.1 Mesh Processing

The following images display how the Mesh processing functions on our add-on, namely, Mesh processor and Split mesh functions alter the active model. After the operators finish executing, we are left with a model that looks something like what's displayed in Figure 34.

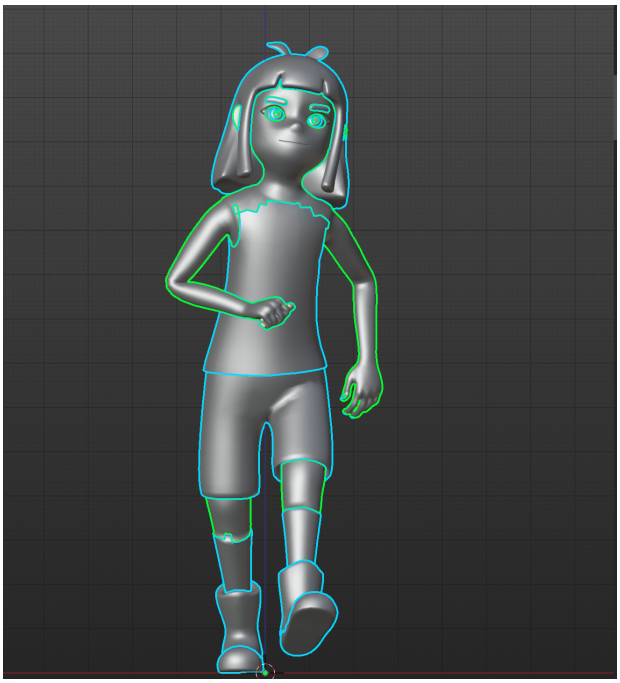

Figure 34: Mixamo model after having been split into colour-based groups - The green and blue outlines select the different partitions.

11.4.2 Pixel Art Renders

Next, the processed mesh has been shaded and split into manageable sections that the user can shade as they like. The images in Figure 35, showcase some pixel art renders of the model, at different levels of detail. These levels can also be selected by the user, depending on how detailed they want the final result to be. For the sake of this example, we chose 2, 5 and 10 scale indexes, as they nicely show the transitioning in the level of detail.

Figure 35: Pixel Art Renders at 2, 5 and 10 Scale indexes

11.4.3 Spritesheet Render

To finish, we include some images of examples of portions of animations exported as spritesheets, which are extremely useful for those who plan on using this add-on for animating characters or other objects in video games. The first subset of frames shows the model of the girl we have been using to exemplify the partial results, happily walking forward. The second one, however, displays another girl running to the side.

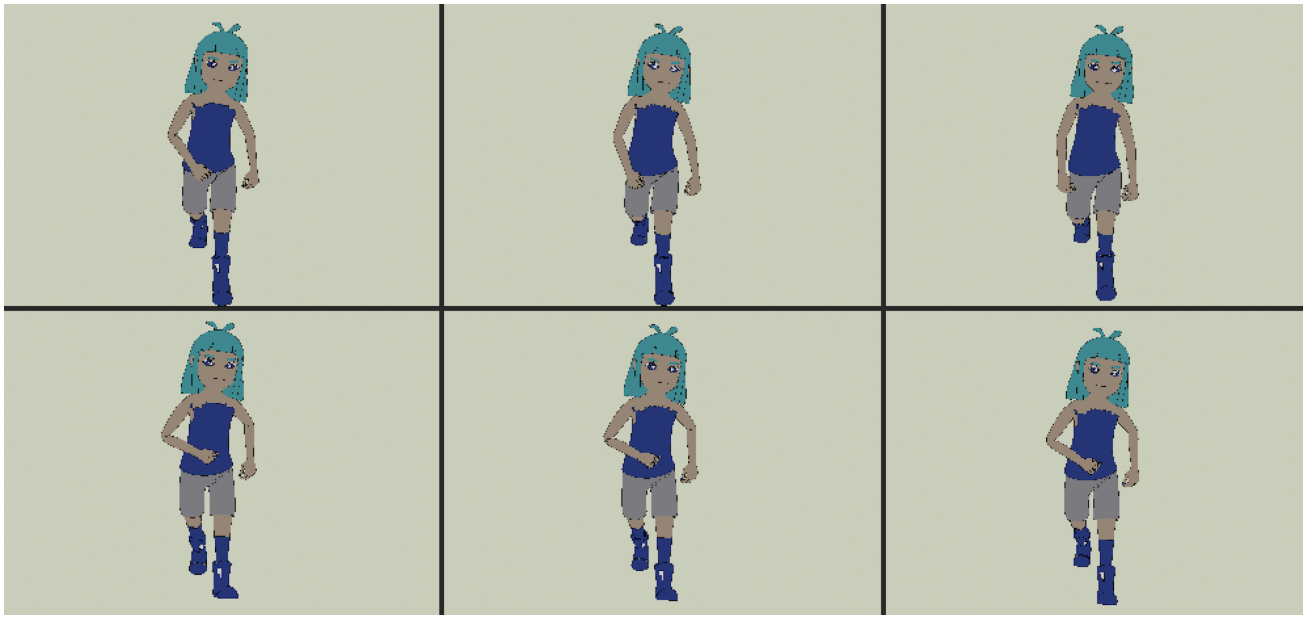

Figure 36: Pixel Art Renders - Spritesheet portion Example 1

Figure 37: Pixel Art Renders - Spritesheet portion Example 2

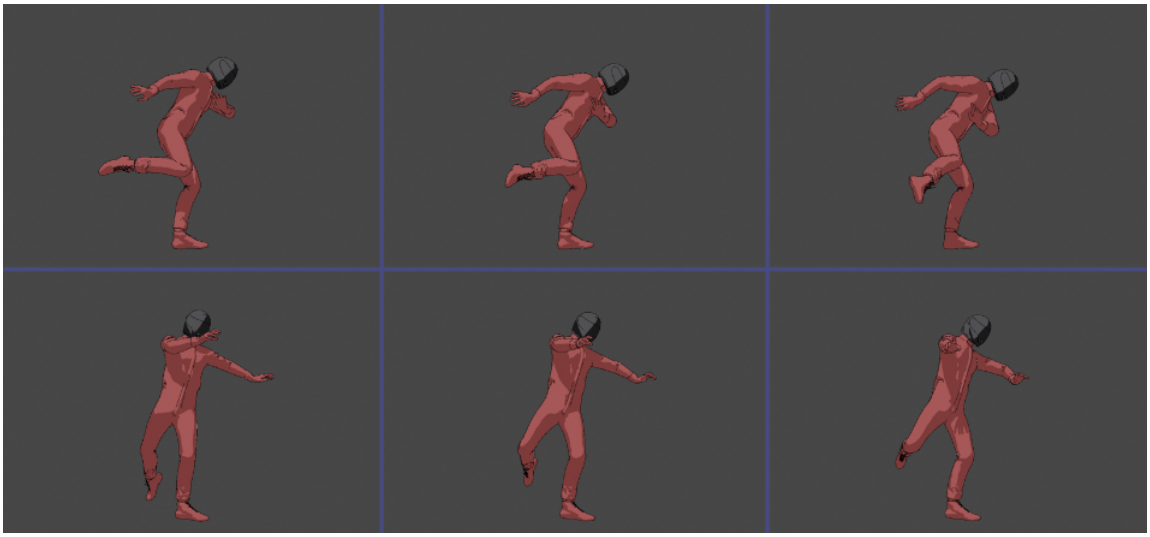

Figure 38: Pixel Art Renders - Spritesheet portion Example 3

12 Conclusions

Overall, this project aimed to develop an automated process for Blender to create 2D animations from 3D models, with the goal of improving upon the existing techniques and reducing the need for manual labour. The project's pipeline involved model creation, shading, compositing, and rendering, with the plugin developed for use within Blender.

As we saw during the literature review sections, the current state of the art in animation production is experiencing a great boom thanks to the appearance of AI-powered tools, including DeepMotion, Cascadeur or MetaHuman Animator. However, research has shown us that there also exist several tools that instead of eliminating the necessity for an animator, enhance their abilities by automating the most clerical aspects of their job. In both cases, however, we found how these tools use machine learning algorithms to analyze motion data and generate realistic 3D meshes, in between frames or natural animations. In addition, both these and the most traditional tools propose several different approaches to tackle the animation process.

In regards to the implementation of the project, it involved the design and development of a plugin for Blender. The plugin was developed with Python, and it aimed to be as efficient as possible, reducing the amount of human interaction required to run the tedious aspects of it. The plugin is also open-source, allowing for greater collaboration and potential contributions from the community. It comes with a unique pipeline that aims to combine the perks of 3D and 2D animation, and successfully bring down the time-effort necessary to put a good animation product together. Additionally, thanks to this automation, it is quite user-friendly, which is something crucial for Blender beginners. It is also true that the project faced several challenges, especially when it comes to the efficiency of the most complex features within the context of Blender. However, this project has still come to a satisfactory conclusion, as we can see from the results obtained.

The results of the project showed that the plugin could help to reduce the time-effort of the animation production process. By automating the process and reducing the need for manual labour, artists can focus on the more artistic aspects of the work, and manage their models in a 3D environment, which can be much more versatile and adaptable than 2D. We have proven that the results obtained are at the level of the available alternatives and that these can be obtained in the same amount of time. In addition, our plugin has the benefit of including three different and widely used styles in a single and simple tool. As a result, this widens the add-on's capabilities, and hence, those of the final user, wether they are professionals, amateurs or beginners.

All in all, the project has the potential to make a significant contribution to the animation industry by improving the workflow for artists, studios, and other stakeholders. The plugin developed in this project represents a step forward in the automation of 2D animation production from 3D models.

13 Future Work

The project has opened up several avenues for future work. While the developed plugin, is already functional and effective in its current state, can be expanded in various ways to increase its utility and applicability. The potential for future work is vast, ranging from the addition of new features and functionalities to the optimization of existing ones. The plugin's open-source nature further facilitates this, allowing developers from around the world to contribute their unique perspectives and ideas. The following sections detail some of the most promising avenues for future work that have been identified.

One of the most exciting prospects is the addition of new art styles. The current implementation focuses on three specific styles of animation: pixel art, cel shading, and traditional 2D. However, the plugin could be expanded to support a wider range of styles. For instance, it could be adapted to generate animations in the style of Japanese anime, or to mimic the aesthetic of classic hand-drawn cartoons. It could also be extended to support more experimental and abstract animation styles, opening up new creative possibilities for artists and animators. This would make it even more versatile and useful to a broader audience of artists and animators.

Keeping the plugin up-to-date with the latest versions of Blender is another important area for future work. As Blender continues to evolve and improve, it is crucial that the plugin remains compatible with the latest versions of the software. This will ensure that users can continue to benefit from the plugin's features and capabilities, regardless of which version of Blender they are using. Additionally, this also means leveraging the newest Blender features, if any were to be applicable and signify a great improvement on the tool's performance or even results.

Publishing the plugin on GitHub and Blender plugin markets is another key step for the future. This will make the plugin available to a wider audience and provide a platform for a community of contributors to form. Open-source software thrives on community contributions, and by making the plugin available on these platforms, we can encourage other developers to contribute their ideas and improvements. This will help to ensure that the plugin continues to evolve and improve over time.

While the project has achieved its initial goals, there is still much work to be done. The future of the project looks bright, with many opportunities for expansion and improvement. We look forward to seeing where these opportunities will lead.

14 Annex A: Planned Tasks - Gantt Chart

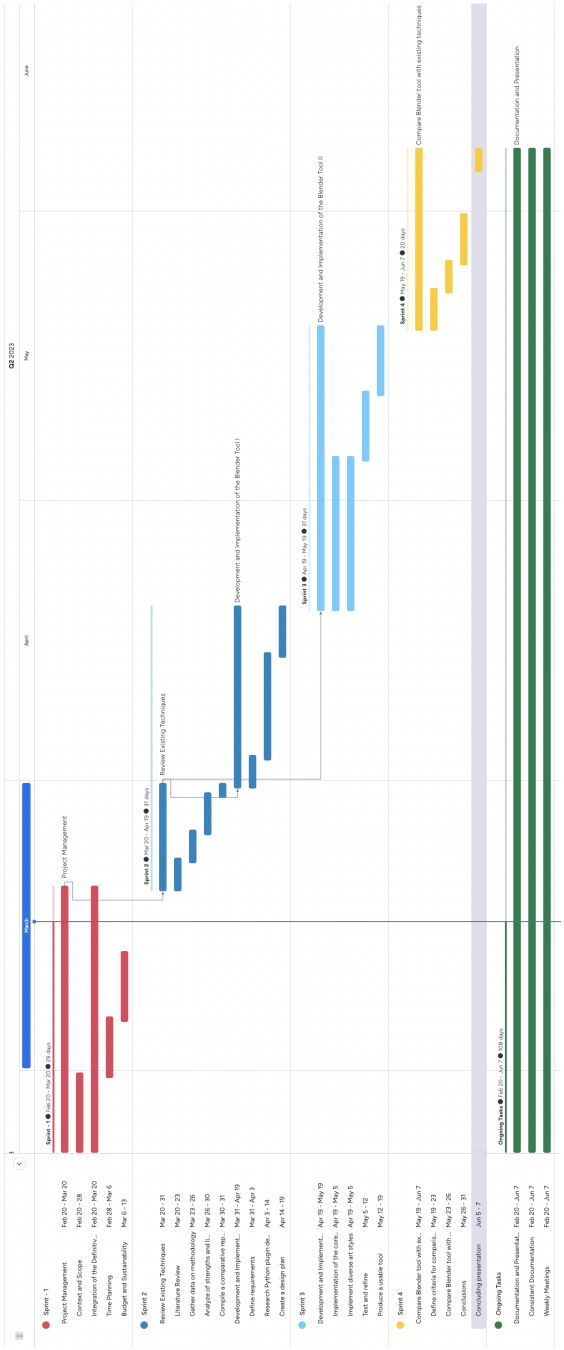

Figure 39: Planned Tasks - Gantt Chart

15 Annex B: Planned Tasks - Updated Gantt Chart

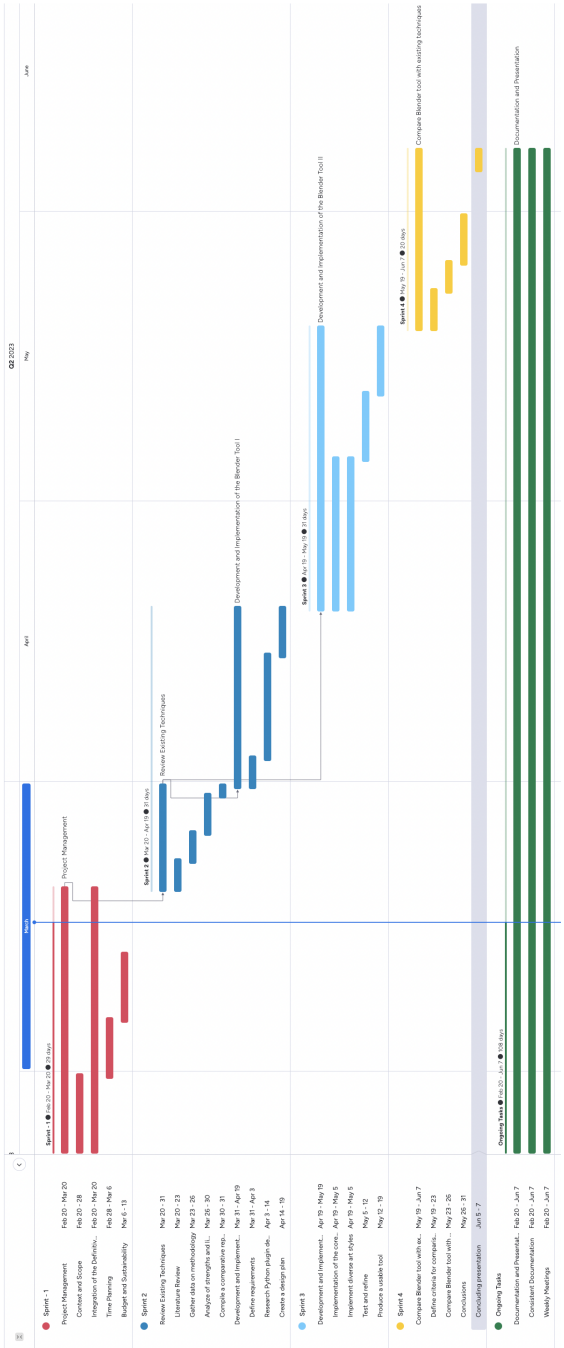

Figure 40: Planned Tasks - Updated Gantt Chart

16 Annex C: Add-on Source code

16.1 Main Panel

```
bl_info = {
    "name": "2D Art Style Renders",
    "author": "Beatriz da Costa",
    "version": (1, 0),
    "blender": (3, 4, 1),
    "location": "View3D > Tool Shelf > 2D Renders",
    "description": "Converts 3D animations to different 2D styles",
}

import bpy
from Operators.OP_mesh_process import MESH_OT_process, SPLIT_OT_mesh
from Operators.OP_tex_simplify import TEXTURE_OT_simplify
from Operators.OP_tex_extract import FBX_TEX_OT_extract
from Operators.OP_style_pixelart import RENDER_OT_PixelArtRender,
    RENDER_OT_PixelArtSpritesheet
from Operators.OP_style_celshading import MATERIAL_OT_CelShading


class RENDER_PT_2DRenders(bpy.types.Panel):
    bl_label = "2D Renders"
    bl_idname = "RENDER_PT_2d_renders"
    bl_space_type = 'VIEW_3D'
    bl_region_type = 'UI'
    bl_category = "2D Renders"

    class RENDER_PT_2DRenders(bpy.types.Panel):
        bl_label = "2D Renders"
        bl_idname = "RENDER_PT_2d_renders"
        bl_space_type = 'VIEW_3D'
        bl_region_type = 'UI'
        bl_category = "2D Renders"

        def draw(self, context):
            layout = self.layout
            scene = context.scene

            # 1. Preprocessing the Mesh Section
            layout.label(text="Mesh preprocessing")

            # Texture subsection
            layout.label(text="Texture")
            box = layout.box()

            # Texture extraction
            box.label(text="Texture Extraction")
            extract_fbx_texture = box.operator(FBX_TEX_OT_extract.bl_idname)
            box.prop(scene, 'output_directory')

            box.separator()

            # Texture simplification
            box.label(text="Texture Simplification")
            box.prop(scene, 'original_texture_path')
            box.prop(scene, 'simplified_texture_path')
            box.prop(scene, 'n_colors')

            simplify_texture_operator = box.operator(TEXTURE_OT_simplify.bl_idname)

            layout.separator()

            # Mesh subsection
            layout.label(text="Mesh")
```

```
62
63              # Mesh processing
64              box = layout.box()
65              box.label(text="Process Mesh")
66              box.operator(MESH_OT_process.bl_idname)
67
68              box.separator()
69
70              # Mesh splitting
71              box.label(text="Split Mesh")
72              box.operator(SPLIT_OT_mesh.bl_idname)
73
74              layout.separator()
75
76              # 2. Cel Shader Style Section
77              layout.label(text="Cel Shader Style")
78              box = layout.box()
79
80              box.prop(scene, "light_color")
81              box.prop(scene, "shadow_color")
82              box.prop(scene, "apply_outline")
83              box.prop(scene, "outline_color")
84              box.prop(scene, "outline_thickness")
85
86              box.operator(MATERIAL_OT_CelShading.bl_idname, text="Apply Cel Shading
                    Material")
87
88              layout.separator()
89
90              # 3. Pixel Art Style Section
91              layout.label(text="Pixel Art Style")
92              box = layout.box()
93
94              box.prop(scene, "pixel_scale", text="Pixel Scale")
95
96              layout.separator()
97
98              # 4. Preview and Render Section
99              layout.label(text="Preview and Render")
100             box = layout.box()
101
102             box.operator(RENDER_OT_PixelArtRender.bl_idname, text="Preview Scene")
103             box.operator(RENDER_OT_PixelArtSpritesheet.bl_idname, text="Render
                    Spritesheet")
104
105
106 def register():
107
108     RENDER_PT_2DRenders.register()
109     RENDER_OT_PixelArtRender.register()
110     RENDER_OT_PixelArtSpritesheet.register()
111     MATERIAL_OT_CelShading.register()
112     TEXTURE_OT_simplify.register()
113     MESH_OT_process.register()
114     FBX_TEX_OT_extract.register()
115     SPLIT_OT_mesh.register()
116     bpy.types.Scene.pixel_scale = bpy.props.FloatProperty(name="Pixel Scale", default
            =1)
117     bpy.types.Scene.light_color = bpy.props.FloatVectorProperty(name="Light Color",
            subtype='COLOR', size=4, min=0, max=1, default=(1, 1, 1, 1))
118     bpy.types.Scene.shadow_color = bpy.props.FloatVectorProperty(name="Shadow Color",
             subtype='COLOR', size=4, min=0, max=1, default=(0, 0, 0, 1))
119     bpy.types.Scene.apply_outline = bpy.props.BoolProperty(name="Apply Outline",
            default=False)
120     bpy.types.Scene.outline_thickness = bpy.props.FloatProperty(name="Outline
            Thickness", default=1, min=0.001, max=10, precision=3)
```

```
121     bpy.types.Scene.shadow_position = bpy.props.FloatProperty(name="Shadow Position",
            default=0.2, min=0.000, max=1, precision=3)
122     bpy.types.Scene.outline_color = bpy.props.FloatVectorProperty(name="Outline Color
            ", subtype='COLOR', size=4, min=0, max=1, default=(0, 0, 0, 1))
123     bpy.types.Scene.output_directory = bpy.props.StringProperty(name="Output
            Directory", description="Path where to save the extracted texture", subtype='
            DIR_PATH')
124     bpy.types.Scene.original_texture_path = bpy.props.StringProperty(name="Original
            Texture Path", description="Path to the original texture", subtype='FILE_PATH
            ')
125     bpy.types.Scene.simplified_texture_path = bpy.props.StringProperty(name="
            Simplified Texture Path", description="Path to save the simplified texture",
            subtype='DIR_PATH')
126     bpy.types.Scene.n_colors = bpy.props.IntProperty(name="Number of Colors", default
            =5, min=1)
127
128 def unregister():
129     RENDER_PT_2DRenders.unregister()
130     RENDER_OT_PixelArtRender.unregister()
131     RENDER_OT_PixelArtSpritesheet.unregister()
132     MATERIAL_OT_CelShading.unregister()
133     TEXTURE_OT_simplify.unregister()
134     MESH_OT_process.unregister()
135     FBX_TEX_OT_extract.unregister()
136     SPLIT_OT_mesh.unregister()
137     del bpy.types.Scene.pixel_scale
138     del bpy.types.Scene.light_color
139     del bpy.types.Scene.shadow_color
140     del bpy.types.Scene.apply_outline
141     del bpy.types.Scene.outline_thickness
142     del bpy.types.Scene.outline_color
143     del bpy.types.Scenes.shadow_position
144     del bpy.types.Scene.output_directory
145     del bpy.types.Scene.original_texture_path
146     del bpy.types.Scene.simplified_texture_path
147     del bpy.types.Scene.n_colors
148
149 if __name__ == "__main__":
150     register()
```

Listing 4: main.py

16.2 Classes

16.2.1 Mesh Processor & Split Class

```
import bpy
from collections import Counter


class MeshProcessor:
    """
    A class that processes a mesh object by assigning vertex groups based on dominant colors from a texture image.
    """

    def __init__(self, obj, texture):
        self.obj = obj
        self.texture = texture

    @staticmethod
    def get_pixel_from_uv(image, uv_coord):
        width, height = image.size
        x = int(uv_coord[0] * width)
        y = int(uv_coord[1] * height)
        pixel_index = (y * width + x) * 4
        pixel_value = (
            int(round(image.pixels[pixel_index] * 255)),
            int(round(image.pixels[pixel_index + 1] * 255)),
            int(round(image.pixels[pixel_index + 2] * 255)),
            int(round(image.pixels[pixel_index + 3] * 255))
        )
        return pixel_value

    @staticmethod
    def calculate_dominant_color(pixel_colors):
        color_counts = Counter(pixel_colors)
        dominant_color = color_counts.most_common(1)[0][0]
        return dominant_color

    def color_distance(self, color1, color2):
        return sum((c1 - c2) ** 2 for c1, c2 in zip(color1, color2))

    def find_closest_color(self, color, colors):
        closest_color = min(colors, key=lambda c: self.color_distance(color, c))
        return closest_color

    def process(self):
        print("Processing mesh...")
        print(f'Number of polygons: {len(self.obj.data.polygons)}')
        print(f'Number of loops: {len(self.obj.data.loops)}')
        print(f'Number of vertex groups: {len(self.obj.vertex_groups)}')

        image = bpy.data.images.load(self.texture)
        uv_map = self.obj.data.uv_layers.active.data

        vg_colors = []
        for vg in self.obj.vertex_groups:
            if vg.startswith('#'):
                rgb = tuple(int(vg.name[i:i+2], 16) for i in (1, 3, 5))
                vg_colors.append(rgb)
        print(vg_colors)

        count = 0
        for poly in self.obj.data.polygons:
            print(f'Processing polygon {count} out of {len(self.obj.data.polygons)}:')
            count += 1
```

```
            pixel_colors = []

            for loop_index in poly.loop_indices:
                loop = self.obj.data.loops[loop_index]
                uv_coord = uv_map[loop.index].uv
                pixel = self.get_pixel_from_uv(image, uv_coord)
                pixel_colors.append(pixel)

                dominant_color = self.calculate_dominant_color(pixel_colors)
                closest_color = self.find_closest_color(dominant_color, vg_colors)
                closest_color_hex = '#' + ''.join(f'{int(x):02x}' for x in
                    closest_color)
                group = self.obj.vertex_groups.get(closest_color_hex) or self.obj.
                    vertex_groups.new(name=closest_color_hex)
                vertices = [self.obj.data.loops[loop_index].vertex_index for
                    loop_index in poly.loop_indices]
                group.add(vertices, 1.0, 'ADD')


class SplitMesh:
    """
    A class that splits a mesh object into separate objects based on color-coded
        vertex groups.
    """

    def __init__(self, obj):
        self.obj = obj

    def split(self):
        print("Splitting mesh...")
        bpy.ops.object.mode_set(mode='EDIT')
        for vg in self.obj.vertex_groups:
            if vg.name.startswith('#'):
                bpy.ops.mesh.select_all(action='DESELECT')
                bpy.ops.object.vertex_group_set_active(group=vg.name)
                bpy.ops.object.vertex_group_select()
                bpy.ops.mesh.duplicate_move(MESH_OT_duplicate={"mode": 1},
                    TRANSFORM_OT_translate={"value": (0, 0, 0)})
                bpy.ops.mesh.separate(type='SELECTED')
        bpy.ops.object.mode_set(mode='OBJECT')
```

Listing 5: mesh_processors.py

16.2.2 Texture Simplifier Class

```python
from PIL import Image
from sklearn.cluster import KMeans
import numpy as np
import os


class TextureSimplifier:
    """
    A class that simplifies textures by quantizing the colors in an image using K-means clustering.
    """
    def __init__(self, image_path, destiny_path, n_colors):
        self.image_path = image_path
        self.destiny_path = destiny_path
        self.image_name, self.image_extension = os.path.splitext(os.path.basename(self.image_path))
        self.n_colors = n_colors

    def quantize_image(self):
        """
            Quantizes the colors in the image using K-means clustering.

            Returns:
                quantized_image (PIL.Image.Image): The quantized image.
                dominant_colors (numpy.ndarray): The dominant colors found by K-means clustering.
        """
        image = Image.open(self.image_path)
        image = image.convert("RGB")

        pixel_data = np.array(image)
        pixel_data = pixel_data.reshape(-1, 3)

        kmeans = KMeans(n_clusters=self.n_colors)
        kmeans.fit(pixel_data)

        quantized_pixel_data = kmeans.cluster_centers_[kmeans.labels_].astype(int)

        quantized_pixel_data = quantized_pixel_data.reshape(image.size[1], image.size[0], 3)

        quantized_image = Image.fromarray(np.uint8(quantized_pixel_data))

        quantized_image.save(self.destiny_path + self.image_name + "__quantized" + self.image_extension)

        return quantized_image, kmeans.cluster_centers_
```

Listing 6: tex_simplifier.py

16.2.3 Pixel Art Compositing Nodes Creator Class

```python
import bpy

class PixelArtNodeCreator:
    """
    A class that creates pixel art nodes in the Compositor for image scaling and pixelation.
    """

    def create_pixel_art_nodes(context):
        scene = context.scene
        scale_value = scene.pixel_scale

        bpy.context.window.workspace = bpy.data.workspaces['Compositing']
        bpy.context.window.view_layer = bpy.context.scene.view_layers[0]

        bpy.context.scene.use_nodes = True
        tree = bpy.context.scene.node_tree
        nodes = tree.nodes

        for node in nodes:
            if node.type not in ['REROUTE', 'OUTPUT_FILE']:
                nodes.remove(node)

        render_layers_node = tree.nodes.new("CompositorNodeRLayers")

        scale_node_1 = nodes.new(type="CompositorNodeScale")
        scale_node_1.label = "Scale (Down)"
        scale_node_1.space = 'RELATIVE'
        scale_node_1.inputs[1].default_value = 1 / scale_value
        scale_node_1.inputs[2].default_value = 1 / scale_value

        pixelate_node = nodes.new(type="CompositorNodePixelate")
        pixelate_node.label = "Pixelate"

        scale_node_2 = nodes.new(type="CompositorNodeScale")
        scale_node_2.label = "Scale (Up)"
        scale_node_2.space = 'RELATIVE'
        scale_node_2.inputs[1].default_value = scale_value
        scale_node_2.inputs[2].default_value = scale_value

        composite_node = tree.nodes.new("CompositorNodeComposite")

        links = tree.links
        render_layers_node = nodes.get("Render Layers")

        links.new(render_layers_node.outputs[0], scale_node_1.inputs[0])
        links.new(scale_node_1.outputs[0], pixelate_node.inputs[0])
        links.new(pixelate_node.outputs[0], scale_node_2.inputs[0])
        links.new(scale_node_2.outputs[0], nodes['Composite'].inputs[0])

        bpy.context.window.workspace = bpy.data.workspaces['Layout']
```

Listing 7: style_pixelart_creator.py

16.3 Operators

16.3.1 Mesh Processing Operators

```python
import bpy
from .Classes.mesh_processors import MeshProcessor, SplitMesh

class MESH_OT_process(bpy.types.Operator):
    """
        Operator class to process a mesh object by assigning vertex groups based on a provided texture file.
    """
    bl_idname = "mesh.process"
    bl_label = "Process Mesh"

    def execute(self, context):
        obj = context.active_object
        texture_file_path = self.file_path
        mp = MeshProcessor(obj, texture_file_path)
        mp.process()
        return {'FINISHED'}

def register():
    bpy.utils.register_class(MESH_OT_process)

def unregister():
    bpy.utils.unregister_class(MESH_OT_process)

class SPLIT_OT_mesh(bpy.types.Operator):
    """
    Operator class to split a mesh object into parts based on the vertex groups it belongs to.
    """
    bl_idname = "split.mesh"
    bl_label = "Split Mesh Per Vertex Groups"

    def execute(self, context):
        obj = context.active_object
        sm = SplitMesh(obj)
        sm.split()

def register():
    bpy.utils.register_class(SPLIT_OT_mesh)

def unregister():
    bpy.utils.unregister_class(SPLIT_OT_mesh)
```

Listing 8: OP_mesh_process.py

16.3.2 Texture Extraction Operator

```python
import bpy
import os

class FBX_TEX_OT_extract(bpy.types.Operator):
    """
    Operator class to extract textures from an FBX file and save them to the
        specified output directory.
    """
    bl_idname = "fbx.extract_textures"
    bl_label = "Extract Textures From FBX"
    bl_options = {'REGISTER', 'UNDO'}

    def execute(self, context):
        obj = bpy.context.active_object
        scene = context.scene
        output_directory = bpy.path.abspath(scene.output_directory)

        for mat in obj.data.materials:
            if mat.use_nodes:
                for node in mat.node_tree.nodes:
                    if isinstance(node, bpy.types.ShaderNodeTexImage):
                        img = node.image
                        if img is not None:
                            output_file = os.path.join(output_directory, img.name + '
                                .png')
                            img.save_render(filepath=output_file)
        return {'FINISHED'}

def register():
    bpy.utils.register_class(FBX_TEX_OT_extract)

def unregister():
    bpy.utils.unregister_class(FBX_TEX_OT_extract)
```

Listing 9: OP_tex_extract.py

16.3.3 Texture Simplifying Operator

```python
import bpy
import os
from Classes.tex_simplifier import TextureSimplifier

class TEXTURE_OT_simplify(bpy.types.Operator):
    """
    Operator class to simplify a texture by quantizing its colors and creating vertex groups for each dominant color.
    """
    bl_idname = "texture.simplify"
    bl_label = "Simplify Texture"
    bl_options = {'REGISTER', 'UNDO'}

    def execute(self, context):
        try:
            obj = context.active_object
            scene = context.scene

            original_texture_path = scene.original_texture_path
            simplified_texture_path = scene.simplified_texture_path
            n_colors = scene.n_colors

            texture_simplifier = TextureSimplifier(original_texture_path, simplified_texture_path, n_colors)
            quantized_image, dominant_colors = texture_simplifier.quantize_image()

            simplified_texture_path = os.path.join(simplified_texture_path, "simplified_texture.png")
            quantized_image.save(simplified_texture_path)

            color_hexes = ['#' + ''.join(f'{int(x):02x}' for x in color) for color in dominant_colors]

            for color_hex in color_hexes:
                obj.vertex_groups.new(name=color_hex)

        except Exception as e:
            print("An error occurred:", str(e))

        return {'FINISHED'}

def register():
    bpy.utils.register_class(TEXTURE_OT_simplify)

def unregister():
    bpy.utils.unregister_class(TEXTURE_OT_simplify)
```

Listing 10: OP_tex_simplify.py

16.3.4 Pixel Art Renderer Operator

```
import bpy
import os
import math
from PIL import Image
from Classes.style_pixelart_creator import PixelArtNodeCreator

class RENDER_OT_PixelArtRender(bpy.types.Operator):
    """
    Operator class to render pixel art by creating pixel art nodes and invoking the default render process.
    """
    bl_idname = "render.pixel_art_render"
    bl_label = "Render Pixel Art"

    def execute(self, context):
        PixelArtNodeCreator.create_pixel_art_nodes(context)
        bpy.ops.render.render('INVOKE_DEFAULT', use_viewport=True)
        return {'FINISHED'}

def register():
    bpy.utils.register_class(RENDER_OT_PixelArtRender)

def unregister():
    bpy.utils.unregister_class(RENDER_OT_PixelArtRender)


class RENDER_OT_PixelArtSpritesheet(bpy.types.Operator):
    """
    Operator class to render a spritesheet by creating pixel art nodes and generating a spritesheet image from the rendered frames.
    """
    bl_idname = "render.pixel_art_spritesheet"
    bl_label = "Render Spritesheet"

    def execute(self, context):
        PixelArtNodeCreator.create_pixel_art_nodes(context)

        scene = context.scene
        frames = scene.frame_end - scene.frame_start + 1
        width, height = scene.render.resolution_x, scene.render.resolution_y
        aspect_ratio = width / height
        max_frames_per_row = 10
        spacing = 10
        spritesheet_width = min(frames, max_frames_per_row)
        spritesheet_height = math.ceil(frames / max_frames_per_row)
        spritesheet_width = min(spritesheet_width, int(spritesheet_height * aspect_ratio))
        spritesheet_height = math.ceil(frames / spritesheet_width)
        spritesheet = Image.new('RGBA', (int((width + spacing) * spritesheet_width - spacing), int((height + spacing) * spritesheet_height - spacing)))

        # Sets the export path for the animation
        export_path = self.export_path
        scene.render.filepath = export_path + "spritesheet_####.png"

        for frame in range(scene.frame_start, scene.frame_end):
            scene.frame_set(frame)
            bpy.ops.render.render(write_still=True)
            image_path = bpy.path.abspath(scene.render.filepath)
            frame_image = Image.open(image_path).convert('RGBA')
            frame_x = frame % spritesheet_width
            frame_y = spritesheet_height - 1 - frame // spritesheet_width
            x = int((width + spacing) * frame_x)
            y = int((height + spacing) * frame_y)
```

```
61                 spritesheet.paste(frame_image, (x, y))
62                 os.remove(image_path)
63 
64             # Saves the spritesheet in the export folder
65             spritesheet_path = export_path + "spritesheet.png"
66             spritesheet.save(spritesheet_path)
67 
68             return {'FINISHED'}
69 
70 def register():
71     bpy.utils.register_class(RENDER_OT_PixelArtSpritesheet)
72 
73 def unregister():
74     bpy.utils.unregister_class(RENDER_OT_PixelArtSpritesheet)
```

Listing 11: OP_style_pixelart.py

16.3.5 Cel Shader Renderer Operator

```
import bpy

class MATERIAL_OT_CelShading(bpy.types.Operator):
    """
    Operator class to apply cel shading material to an object, along with an optional
        outline material.
    """
    bl_idname = "material.cel_shading"
    bl_label = "Apply Cel Shading Material"
    bl_options = {'REGISTER', 'UNDO'}

    def execute(self, context):
        obj = context.object
        scene = context.scene

        light_color = scene.light_color
        shadow_color = scene.shadow_color
        apply_outline = scene.apply_outline
        shadow_position = scene.shadow_position
        outline_thickness = scene.outline_thickness
        outline_color = scene.outline_color

        cel_shading_mat = bpy.data.materials.new(name="Cel_Shading_Material")
        cel_shading_mat.use_nodes = True
        nodes = cel_shading_mat.node_tree.nodes
        links = cel_shading_mat.node_tree.links

        for node in nodes:
            nodes.remove(node)

        diffuse_node = nodes.new(type="ShaderNodeBsdfDiffuse")
        shader_to_rgb_node = nodes.new(type="ShaderNodeShaderToRGB")
        color_ramp_node = nodes.new(type="ShaderNodeValToRGB")
        color_ramp_node.color_ramp.interpolation = "CONSTANT"
        color_ramp_node.color_ramp.elements[1].color = light_color
        color_ramp_node.color_ramp.elements[0].color = shadow_color
        color_ramp_node.color_ramp.elements[1].position = shadow_position
        material_output_node = nodes.new(type="ShaderNodeOutputMaterial")

        links.new(diffuse_node.outputs[0], shader_to_rgb_node.inputs[0])
        links.new(shader_to_rgb_node.outputs[0], color_ramp_node.inputs[0])
        links.new(color_ramp_node.outputs[0], material_output_node.inputs[0])

        obj.data.materials.clear()
        obj.data.materials.append(cel_shading_mat)

        if apply_outline:
            outline_mat = bpy.data.materials.new(name="Outline_Material")
            outline_mat.use_nodes = True
            nodes = outline_mat.node_tree.nodes
            links = outline_mat.node_tree.links

            for node in nodes:
                nodes.remove(node)

            emission_node = nodes.new(type="ShaderNodeEmission")
            emission_node.inputs[0].default_value = outline_color
            material_output_node = nodes.new(type="ShaderNodeOutputMaterial")

            links.new(emission_node.outputs[0], material_output_node.inputs[0])

            outline_mat.use_backface_culling = True

            solidify_mod = obj.modifiers.get("Solidify_Outline")
```

```
64              if not solidify_mod:
65                  solidify_mod = obj.modifiers.new(name="Solidify_Outline", type="
                        SOLIDIFY")
66
67              solidify_mod.thickness = outline_thickness
68              solidify_mod.use_flip_normals = True
69              solidify_mod.material_offset = 1
70
71              obj.data.materials.append(outline_mat)
72
73          return {'FINISHED'}
74
75  def register():
76      bpy.utils.register_class(MATERIAL_OT_CelShading)
77
78  def unregister():
79      bpy.utils.unregister_class(MATERIAL_OT_CelShading)
```

Listing 12: OP_style_celshader.py

17 Bibliography & References

[1] [1] "Animation," Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Animation.

[2] "Interpolation, easing, and smoothing," Webflow University. [Online]. Available: https://university.webflow.com/lesson easing-and-smoothing.

[3] "Motion capture and animation," Adobe Creative Cloud. [Online]. Available: https://www.adobe.com/uk/creativeclou capture.html.

[4] "Artificial intelligence," Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Artificial-intelligence.

[5] D. Eran, "The Filmmaker's Guide to Visual Effects: The Art and Techniques of VFX for Directors, Producers, Editors and Cinematographers," Focal Press, 2017.

[6] "Blender - a 3D creation suite," Blender Foundation. [Online]. Available: https://www.blender.org/.

[7] "Community," Blender Foundation. [Online]. Available: https://www.blender.org/community/.

[8] "3D Motion to 2D Animation," Reallusion. [Online]. Available: https://www.reallusion.com/cartoon-animator/3d-motion-to-2d-animation.html.

[9] "Mixamo," Adobe Inc. [Online]. Available: https://www.mixamo.com/.

[10] "DeepMotion - AI-Powered Motion Synthesis and Simulation," DeepMotion Inc. [Online]. Available: https://www.deepmotion.com/.

[11] "Cascadeur - Physics-based animation software for 3D professionals," Cascadeur LLC. [Online]. Available: https://cascadeur.com/.

[12] T. Lark, "Character's Pose Prediction using 6 Points in Cascadeur," Cascadeur LLC. [Online]. Available: https://oldforum.cascadeuhttps://www.salary.com/r.com/articles/160-character%E2%80%99s-pose-prediction-using-6-points-in-cascadeur.

[13] "Linear - The Issue Tracking Tool You'll Enjoy Using," Linear Labs Inc. [Online]. Available: https://linear.app/.

[14] "Indeed," Indeed. [Online]. Available: https://es.indeed.com/?from=gnav-title-webapp.

[15] "Salary.com," Salary.com. [Online]. Available: https://www.salary.com/.

[16] "GitHub," GitHub. [Online]. Available: https://github.com/.

[17] "3D Viewport Shading," Blender Manual. [Online]. Available: https://docs.blender.org/manual/en/2.79/editors/3dvie

[18] "Compositing," Blender Manual. [Online]. Available: https://docs.blender.org/manual/en/latest/compositing/index.h

[19] "Devicers," Devicers. [Online]. Available: https://www.devicers.com/

[20] "Gotcha's," Blender API Manual. [Online]. Available: https://docs.blender.org/api/current/info$_g otcha.html strange-errors - using - threading - module$"$Persistence of vision$,"$Wikipedia.[Online].Available : https : //en.wikipedia.org/wiki/Persistence$_o f_v ision$

[21] "Rotoscoping in Animation," Adobe Creative Cloud. [Online]. Available: https://www.adobe.com/creativecloud/video/disco animation.html

[23] "Security in Adobe Animate," Adobe Help Center. [Online]. Available: https://helpx.adobe.com/la/security/products/anima

[24] "Disney Animation: The Illusion of Life," Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Disney$_A nimation : T$ $he I llusion_o f_L ife$"$Unreal Engine$,"$Unreal Engine.[Online].Available : https : //www.unrealengine.com/en - US$

[26] "Computer Science Information Technology," AIRCC Publishing Corporation. [Online]. Available: https://aircconline.com/

[27] "SCITEPRESS Digital Library," SCITEPRESS. [Online]. Available: https://www.scitepress.org/Papers/2022/108151/10815

[28] "arXiv.org," Cornell University. [Online]. Available: https://arxiv.org/abs/2209.00185

[29] "De Gruyter," De Gruyter. [Online]. Available: https://www.degruyter.com/document/doi/10.1515/comp-2022-0255/html

[30] "Maya — Computer Animation Modeling Software," Autodesk. [Online]. Available: https://www.autodesk.com/products/m
YEARtab=subscription

[31] "ZBrush," Pixologic. [Online]. Available: https://pixologic.com/zblanding/?v=2

[32] "Aseprite," Aseprite. [Online]. Available: https://www.aseprite.org/

[33] "Piskel," Piskel. [Online]. Available: https://www.piskelapp.com/

[34] "Cinema 4D," Maxon. [Online]. Available: https://www.maxon.net/es/cinema-4d

[35] "Pixel Art — Basics Tutorial," Lucas Roedel on Gumroad. [Online]. Available: https://lucasroedel.gumroad.com/l/pixel$_art$