Doctoral Thesis

Generation of 3D Characters from Existing Cartoons and a Unified Pipeline for Animation and Video Games

Simone Barbieri

Centre for Digital Entertainment
Faculty of Media and Communication, Bournemouth University

September 2020

Simone Barbieri: *Generation of 3D Characters from Existing Cartoons and a Unified Pipeline for Animation and Video Games*

SUPERVISORS:

Dr Xiaosong Yang (Bournemouth University)
Dr Zhidong Xiao (Bournemouth University)
Ben Cawthorne (Thud Media)

Abstract

Despite the remarkable growth of 3D animation in the last twenty years, 2D is still popular today and often employed for both films and video games. In fact, 2D offers important economic and artistic advantages to production. In this thesis has been introduced an innovative system to generate 3D character from 2D cartoons, while maintaining important 2D features in 3D as well. However, handling 2D characters and animation in a 3D environment is not a trivial task, as they do not possess any depth information.

Three different solutions have been proposed in this thesis. A 2.5D modelling method, which exploits billboarding, parallax scrolling and 2D shape interpolation to simulate the depth between the different body parts of the characters. Two additional full 3D solution have been presented. One based on inflation and supported by a surface registration method, and one that produces more accurate approximations by using information from the side views to solve an optimization problem.

These methods have been introduced into a new unified pipeline that involves a game engine, and that could be used for animation and video games production. A unified pipeline introduces several benefits to animation production for either 2D and 3D content. On one hand, assets can be shared for different productions and media. On the other hand, real-time rendering for animated films allows immediate previews of the scenes and offers artists a way to experiment more during the making of a scene.

Acknowledgements

I am deeply grateful to my supervisors Dr Xiaosong Yang and Dr Zhidong Xiao, for the tremendous support throughout all these years. Their help has been invaluable to carry on this path. Among all the meetings, I will never forget the first interview for the EngD with Dr Yang back in 2015. Thank you for believing in me.

I would like to express my sincere gratitude to Zoe Leonard, Mike Board and all the people at the Centre for Digital Entertainment and Bournemouth University that assisted me at every stage of my project.

I would like to extend my sincere thanks to Ben Cawthorne, Jon Rennie and everyone at Thud Media and Cloth Cat Animation. Especially Kim, Aled and Josh, your help and experience have been precious for completing this thesis.

I would also like to thank Prof Christos Gatzidis and Prof Feng Dong for their review of the thesis and for providing suggestions to elevate this project.

A gigantic thanks to my parents and my family for pushing me forward since day one. I want to offer my special thanks to Marco for his neverending friendship and support. And a thank you to all the friends that, for all these years, have been close to me, whether physically or virtually.

Last but definitely not least, thanks to my wife, Angelica, foolish enough to marry me during the doctorate. Thank you for keeping me from going insane during all these years, particularly during the lockdown and while I was trying to complete this thesis. Without you, I would have probably dropped it a million times.

Declaration

This thesis has been created by myself and has not been submitted in any previous application for any degree. The work in this thesis has been undertaken by myself except where otherwise stated.

Contents

List of Publications

- Poster to **SIGGRAPH 2016**:

  Barbieri, S., Garau, N., Hu, W., Xiao, Z., and Yang, X. [2016]. Enhancing character posing by a sketch-based interaction. In *ACM SIGGRAPH 2016 posters on - SIGGRAPH '16*. ACM Press.

- Paper in **Lecturer Notes in Computer Science**:

  Barbieri, S., Cawthorne, B., Xiao, Z., and Yang, X. [2017]. Repurpose 2D character animations for a VR environment using BDH shape interpolation. In *ACM Next generation computer animation techniques* (pp. 69–85). Springer International Publishing.

- Poster to **SIGGRAPH 2018**:

  Barbieri, S., Jiang, T., Cawthorne, B., Xiao, Z., and Yang, X. [2018]. 3D content creation exploiting 2D character animation. In *ACM SIGGRAPH 2018 posters on - SIGGRAPH '18*. ACM Press.

List of Figures

Glossary

AR	Augmented Reality
CNN	Convolutional Neural Network
DAG	Direct Acyclic Graphs
FBX	**F**ilm**b**o**x**
FEM	Finite Element Method
fps	Frames per Second
glTF	Graphics Library Transmission Format
JSON	JavaScript Object Notation
PDE	Partial Differential Equation
traditional platforms	all those conventional environments which run software or reproduce media through a screen
Triple-A	**A** lot of time, **A** lot of resources, **A** lot of money
USD	Universal Scene Descriptor
VFX	Visual Effects
VR	Virtual Reality

Chapter 1

Introduction

1.1 Background

During the 80s, the continuous growth of commercial hardware, in particular of graphic workstations, together with the advancements in computing power, brought to an expansion of 3D graphics in the entertainment industry. 3D graphics evolved further in the 90s, with the release of the first feature film fully computer-animated, *Toy Story* [Lasseter et al., 1995]. Despite the growth of 3D, 2D animation is still often employed for both films and video games.

For what concerns the animation industry, while mainstream western productions have moved almost entirely to 3D, other areas of the world are still firmly focused on traditional 2D animation. Japanese animation is a case in point, with *Studio Ghibli* as one of the most prestigious examples, renowned all worldwide for producing films the likes of *Spirited Away* [Miyazaki, 2001]. In the west, 2D animation is more common in TV productions. *The Simpsons* [Groening et al., 1989] is one of the most enduring animated TV shows, running since 1989.

Regarding video games, while Triple-A publishers — **A** lot of time, **A** lot of resources, **A** lot of money [DeMaria and Wilson, 2003] — focuses on the production of 3D games, 2D games are being developed mainly through independent studios [Kay, 2018]. Independent video game developers, in fact, can afford to risk more, as they are outside market dynamics that Triple-A companies must undertake, and their production involves a significantly lower economic investment [Morris, 2018; Kruger, 2018]. This kind of productions, the indie video games, in recent years have spread widely and have become an important reality in the video game industry [Lipkin, 2012], and a large number of these games are realised in 2D. The reason behind it could

be either for an aesthetic choice, such as in *Cuphead* [Moldenhauer and Moldenhauer, 2017], or to reduce the costs. In fact, it is common that only a few people work on these games, or, in some cases, even single individuals, such as *Braid* [Blow, 2008] or *Undertale* [Fox, 2015].

In both films and video games, often 2D and 3D elements are combined to obtain effects that in 2D — or 3D — alone could not be achieved as easily. In films, one of the first examples is *Beauty and the Beast* [Trousdale et al., 1991], where, in the famous ballroom scene, the 2D characters are combined with a 3D background, allowing the animators to handle the 3D space, depth and shading of the scene in a smoother way [Haswell, 2014]. Another example could be seen in the film *Who Framed Roger Rabbit* [Zemeckis et al., 1988], in which cartoon characters interact with the real world. In video games, the billboarding rotates 2D elements in the 3D environment towards the camera to prevent the user from seeing that they are 2D. This technique is used primarily for user interfaces but also for effects, background elements — trees, clouds — or characters. For instance, it is employed in *Mario Kart 64* [Konno and Miyamoto, 1996], where the characters are prerendered from 3D models, but they appear as 2D sprites in the actual game.

1.2 Research Aim

This research project aims to employ existing 2D cartoon character animations in 3D by generating a 3D approximation of the character. In particular, the focus is on maintaining all its distinctive traits in 3D, which makes them highly recognisable and provides the artists with more expressive freedom. It also aims to generate 3D animations from 2D ones, to repurpose existing 2D animations from previous productions as well. These new techniques have been integrated into a new, unified pipeline, designed to be used for both animation and video game production, that involves the employment of a game engine.

1.3 Thud Media and Cloth Cat Animation

This research project is the central work carried out during an Engineering Doctorate (EngD) in Digital Media, which has been held in cooperation between the Centre for Digital Entertainment (CDE) from Bournemouth University and **Thud Media**[1]. Thud Media is an interactive design and development studio which makes games, apps and websites for iOS, Android and

[1] http://thudmedia.com/

browser platforms. They develop standout interactive content and experiences, animated games for all ages, including specialised preschool content. They build websites that meet high design expectations and have created interactive eBooks, quiz show apps, second screen experiences and more for a variety of projects and clients. Thud Media is a member of Games Wales[2], a non-profit industry group made up of Welsh games developers, educational institutions, media partners and industry bodies with a shared interest in promoting the games industry in Wales.

Thud Media is based in Cardiff, alongside its sister company **Cloth Cat Animation**[3], an animation studio with expertise in the development, design and technical execution of content for broadcast series, commercials, games and web. They have worked on a wide range of 2D and 3D projects for clients all over the world, and their focus is on creating high-quality character animation for all age groups. Their projects include *Luo Bao Bei* (International Emmy nomination 2019), *Shane the Chef*, *Olobob Top*, *The Rubbish World of Dave Spud*, *Toot the Tiny Tugboat* and the award-winning *Ethel & Ernest*, which was released in cinemas in 2016. They have collaborated on multiple projects with international partners and are one of the few studios able to handle a series from development to platform delivery.

1.3.1 Impact

Thud Media is Cloth Cat Animation's interactive publisher and developer and has collaborated with them to produce apps and games for a number of their animation series. From the two companies' perspective, the objective was to employ the characters of their animated series in video games and VR experiences. There were two main challenges to solve to achieve this objective.

- The employment of the 2D cartoon characters in a 3D environment. To use those characters in 3D, the artists were required to model every character in 3D from scratch.
- Moving the assets from one pipeline to the other. Even for 3D animated series, it required to reimport every model and animation and entirely remake the materials, the lighting and the layout of each scene.

These two operations were challenging in particular for their time-consuming component. The impact of the research presented in this thesis influences both these aspects of the companies' production.

[2]http://gameswales.org/
[3]http://clothcatanimation.com/

For what concerns the time required to model a 3D character from scratch, it can vary considerably, depending on the complexity of the character. In the previous 3D animated series produced by Cloth Cat, *Shane the Chef*, modelling a cartoon character of medium complexity took on average twenty hours of work, without any texturing, retopology and not considering the time for feedback and changes. The proposed system can drastically reduce the required amount by creating a close approximation of that 3D character from the 2D drawings, leaving to the artists only the refinements, which takes on average about three hours, again, depending on the complexity. *Shane the Chef* has about twenty characters. With an average of seventeen hours saved per character on modelling, this system allows the studio to save hundreds of hours of work. The same applies to the time necessary for a 3D animation. For the ten minutes episodes of *Shane the Chef*, the animators had a scene each to block and to animate, which took on average seven days. With the proposed system, the animators need only to refine the animation automatically generated, which takes on average five hours.

Regarding the second aspect, moving the assets from the animation pipeline to the video game development one is a complex operation. In fact, because of the size of an average scene for either an animated series or a video game, building one from the ground up takes several weeks. Although the models are the same, it is not always possible to export the scene's layout, so it is usually required to rebuild the scene entirely. Moreover, every software handles materials and lighting differently, so they must be adjusted for every single asset. The new pipeline, by involving a game engine, allows the companies not only to use the same tools to realise and animate the character but directly to share entire scenes for both the animated series and the games, thus saving a significant amount of time and resources. This also permits the parallelisation of the work so that the series and the games are developed simultaneously. Additionally, the inclusion of a game engine in the pipeline introduces real-time rendering in animation production, allowing to save not only more time but also more creativity and experimentation.

The integration of 2D content in a 3D environment opens the door even to more possibilities. For instance, the introduction of 2D characters or elements in Virtual Reality (VR) or Augmented Reality (AR) experiences. In traditional platforms — all those conventional environments which run software or reproduce media through a screen — often 2D represents a more economical and immediate alternative to the 3D animation. Introducing 2D characters to VR could bring the same advantages it has in traditional platforms, incentivising creators to produce more content for this platform and avoiding it to become merely a gimmick or a tech demo [Scherba, 2016].

Figure 1.1: Mickey's ears appear circular from any view [Sporn, 2009].

1.4 Motivation

While 3D graphics allows the creation of diverse kinds of application compared to 2D, including the involvement of popular technologies such as VR or AR, 2D still has several advantages:

- 2D animations are faster to produce, having a dimension less to consider.
- 2D assets are more rapid to realise.
- The data sizes for the 2D assets, compared to the 3D ones, are smaller and thus easier to maintain.
- 2D applications perform better, even on low-end devices, which imply a more significant market share.

However, 2D is used not only for economic reasons but also as a specific artistic direction. Indeed, 2D characters have very distinctive traits, which often are lost when transposed to 3D. An iconic example is Mickey Mouse, whose ears appear circular no matter which way he is facing [Sporn, 2009], as shown in figure 1.1.

The automatic generation of 3D characters from 2D hand-drawn cartoon characters is not straightforward. While there are several solutions to generate a 3D model from two or more photographs from different points of view of a physical object, the same is not valid for cartoons. In fact, two fundamental reasons prevent the employment of these techniques with 2D sketches.

First, the fact that a cartoon is hand-drawn entails a lack of accuracy between the different perspectives from which the character is represented. There might be inconsistencies between two different sketches of the same character because of the hand-made nature of the medium, and these are accentuated when the character is drawn from different perspectives. The differences, in some cases, are even wanted by the artists, as often cartoons are exaggerated, and their representation might differ significantly from one

perspective to the next, again, as illustrated in figure 1.1. These differences include those specific traits discussed earlier, which this research aims to maintain in 3D. Secondly, the methods of 3D reconstruction from photos make use of minor details present in the picture to find the same points from multiple perspectives and use those to recreate the 3D model. The way cartoons are coloured and their lack of details make it impossible to employ methods of 3D reconstruction such as those used for photos.

The problem is approached in two different ways. While the first solution attempted to keep some of the elements in 2D, with a 2.5D approach, the second proposal is fully 3D, with two variants. The first one is based on inflation. To keep the 2D traits of the characters, the method generates the inflated model for each sketch of the turnaround, and it employs a surface registration method to interpolate between the different models while the camera rotates around the character. Instead, the second variant of the 3D approach uses information from the side views of the turnaround to generate a 3D model that directly incorporates the distinctive traits by solving an optimisation problem.

These methods are accompanied by a novel method to generate a 3D animation from a 2D one, with the main objective of repurposing the existing 2D animations made for previous productions. This method actually employs a sketch-based posing system, treating the 2D skeleton poses of the 2D animations as sketches. There are two main problems to solve when using sketches to pose a 3D skeleton: finding the correspondence between the sketch and the model and compute the depth, absent in a 2D drawing. Two interlocking optimisation problems have been formulated to approach these issues: a linear assignment discrete problem on the correspondence and a non-linear continuous problem for the deformation.

These new techniques are introduced in a new, unified pipeline designed to be used for the production of both animation and video game production by involving a game engine. In the entertainment industry, the animation pipeline is defined primarily by the medium for which the animation will be made. For instance, the pipeline for a 2D film or a 3D one would be very different, and, in turn, it would be different from the one for a 2D video game.

The employment of a different pipeline depending on the context is comprehensible and widespread, as the different kinds of animations have different requirements that, historically, were incompatible. Video games require a constant frame rate of at least 30 fps — Frames per Second — to be enjoyable, at the cost of an inferior graphics rendering. Animation for film, on the contrary, needs the best graphic rendering possible, with slow rendering times. For instance, the rendering of the film *Monsters University* took 29 hours per frame [Takahashi, 2013]. For this reason, artists often employ pre-visualisation

tools, which offer an approximated preview of the scene, usually based on the storyboard, before they start to work on it.

Nowadays, machines remarkably evolved since these pipelines have been established. With this higher computing power, computers can render extremely high graphic quality scenes in real-time with the game engines. Reportedly, real-time rendering for studios could represent a significant economic and time saving [Bąk and Wojciechowska, 2019a]. The real-time rendering provided by game engines offers crucial support during the realisation of a film, allowing the filmmakers to preview the scenes almost instantly and close to the final product in terms of visual quality [Ramos, 2017]. These instruments allow the production team to experiment more during the making of the scenes.

Despite the advantages of the employment of game engines in animation production, introducing them in the animation pipeline is not a simple process. Software like Blender or Maya, the most common choices for animation production, are all-in-one solutions that include tools for modelling, rigging, animation, and most of the animation pipeline steps. On the other hand, game engines handle only a few steps of the pipeline, particularly the layout, the lighting, the physics, and the rendering. They do not have any tool for modelling, rigging or animation, and all these assets have to be imported from external software. While this is common in video game development, it is not for animation.

This project introduces a new communication layer to tackle this problem in the pipeline that extends to every tool employed. The objective of this communication layer is to handle saving and loading the assets from any software involved in the pipeline, making them accessible from any other software, in order to avoid a continuous export and import of assets. The communication layer also handles versioning so that the artists can load any version of a specific asset and also change the version of any asset in the scene dynamically. For the purpose of this thesis, Blender has been used as modelling, rigging and animation tool, and Unreal Engine has been used as game engine. However, the pipeline has been designed to be modular so that any tool can be integrated and any game engine could be used.

There are several examples of short films realised with a game engine. Unity, for instance, has been used for several of these. *Adam* [Efremov, 2016] is one of the most remarkable examples; the Unity Technologies Team has realised it to show the potentiality of their game engine in the filmmaking field. Adam won several prizes, and after its success, Neill Blomkamp, writer and director of *District 9*, *Elysium* and *Chappie*, was employed with his independent studio *Oats Studios* to produce two more episodes of the story. *The*

Gift [Kajisa, 2016] and *Ultimate Bowl 2017* [Kajisa, 2017], from Marza Animation Planet, are also made with Unity. Even Disney announced a short series that uses real-time rendering made with Unity [Amidi, 2018], *Baymax Dreams*, continuing their partnership after Disney's first VR short film, *Cycles* [Wells, 2018]. Other recent short films made with Unity include *The Heretic* [Efremov, 2019] and *Love & 50 Megatons* [Schick, 2020].

Unreal Engine is also being used to produce animated films. Few examples are the feature film *Allahyar and the Legend of Markhor* [Khan, 2018] and the short film *A Boy and his Kite* [Antoine et al., 2015]. Moreover, the droid K-2SO, a character in *Star Wars: Rogue One*, has been rendered in real-time using Unreal Engine [Alexander, 2017]. More recently, this engine has been used to make a photo-real set for *The Mandalorian* [Farris, 2020]. Using 6 meters high, 270 degrees semi-circular screens, the team could place and control the environment and effects in real-time. This has several advantages during the production, including the actors that were able to react to events happening in front of them and the accurate lighting and reflections for the complex VFX shots.

There has been a significant commitment from the companies to get their game engines ready for filmmaking. Epic is encouraging artists to use Unreal to produce films [King, 2015] by setting it royalty-free for any non-interactive storytelling purpose, a move that could also help independent filmmakers [Takahashi, 2017]. Likewise, the Unity team is releasing numerous tools for film production in the newest releases of their software [Muller, 2017; Savov, 2017].

1.5 Research Questions

Considered the motivations and the problems expressed in section 1.4, this thesis raises the following questions.

Research Question 1: *Is there a suitable method to represent an approximation of existing 2D cartoon characters in 3D?*

Research Question 2: *Is it possible to preserve essential and specific 2D features in 3D?*

Research Question 3: *Does the generation of approximated 3D models and animations from existing 2D cartoon characters have a benefit on the production?*

Research Question 4: *Can a game engine be integrated into the production pipeline for animation?*

Research Question 5: *Is there a benefit in the integration of a game engine in the animation pipeline?*

The first three questions are discussed in chapter 3, while the last two in chapter 4. A summary of the answers to the questions is provided in chapter 5.

1.6 Contributions

This thesis provides the following contributions:

Contribution 1: We propose a new method for generating 2.5D characters based on billboarding, parallax scrolling and 2D shape interpolation to simulate the depth between the different body parts of the characters. Compared to other 2.5D modelling methods, the one proposed in this thesis supports existing 2D cartoon sketches as input.

Contribution 2: We propose two new methods for generating 3D characters from 2D cartoon sketches, designed to keep specific traits typical of 2D cartoon characters in 3D.

The first one is based on inflation and supported by a non-rigid surface registration method. An inflated model is generated for each perspective of the turnaround of the character and computes the registration between the models of adjacent perspective. To make the distinctive elements visible in the 3D version of the character, then, while the camera rotates around the character, the vertices are interpolated from one position to the other. The second method, instead, uses the information from the side view sketches and, by solving an optimisation problem, generates a 3D model that keeps the specific traits of the 2D character directly.

Contribution 3: We propose a new method to generate a 3D animation from an existing 2D one. This method is based on a sketch-based posing method, and it treats the skeleton for each key-frame of the animation as a 2D sketch to pose the 3D skeleton.

Contribution 4: We propose a new pipeline that can be used for both animation and video game production. This pipeline involves the use of a game engine and has two main advantages compared to a traditional pipeline. First, it

introduces the real-time rendering in animation production, thus making the production faster, as it does not require to render the entire scene every time to check on potential problems, and more creative, as the artists can experiment more during the production of the animation.

1.7 Thesis Outline

The thesis is structured as follow. The first part on **Research Background** consists of two chapters.

- Chapter 1 outlines the background of the problem that this thesis aims to solve. It introduces the companies that supported the research project and its impact on the industry.
- Chapter 2 offers a literature review of the topics related to the methodology employed to answer the Research Questions listed in section 1.5.

The second part includes two chapters as well and describes the **Methodology** used to solve the problems stated in section 1.4 and to achieve the contributions in section 1.6.

- Chapter 3 focuses on the generation of the 2.5D and 3D approximations of models and animations from existing 2D sketched cartoon characters. This chapter is divided into several sections.
  - Section 3.1 explains the methodology to generate a 2.5D model from existing 2D cartoon characters created during the animation production, which corresponds to contribution 1.
  - Section 3.2 provides two different methods for generating 3D models from the 2D cartoon characters, which represents contribution 2. One is based on inflation, in section 3.2.1, and the other on an optimisation method, in section 3.2.2.
  - Section 3.3 introduces a method to pose 3D characters from 2D sketches. This method is then used to generate a whole animation for 3D characters from their 2D version. This corresponds to contribution 3.
  - Section 3.4 contains an analysis of the results.
- Chapter 4 introduces the proposed pipeline for animation and explains every step in detail while also presenting the system developed for the companies, which represents contribution 4.

The final part, **Concluding Remarks**, contains a final chapter that summarises the contribution, answers the questions raised in section 1.5 and closes the thesis outlining possible future works.

Chapter 2

Literature Review

This thesis focuses on two main topics. The first one is the generation of 3D characters from 2D cartoons. The reconstruction of 3D models from 2D images is a broad and well-studied topic. This thesis, however, focuses only on the generation of 3D characters from 2D cartoons. In fact, the main objective of the thesis is the repurposing of 2D animated cartoon characters in 3D. Cartoons and sketches, differently from other kinds of 2D images such as photographs, cannot rely on the accuracy, because of their hand-drawn nature.

To better understand the complexity of the problem, this chapter provides an analysis of the current state of the art for sketch-based methods for modelling and animation. Section 2.1 starts with the analysis of 2.5D modelling methods and the concept of *layering*, while section 2.2 and section 2.3 introduce sketch-based modelling and animation respectively. Additionally, section 2.4 analyses recent sketch-based works that employ deep learning and gives an explanation on why this approach, which is becoming increasingly popular in all the branches of computer science research, has not been followed in this thesis.

The second topic of the thesis, instead, is a unified pipeline that can be used for animation and video game production. Section 2.5, to have a better comparison with the new pipeline introduced in this thesis, describes a typical pipeline for 2D and 3D animation and analyses the main difference with a video game pipeline. Moreover, this section describes other recent pipelines that employ a game engine and compares them with the one proposed in this thesis.

This review highlights the evolution of the field, shows the main limitations of the current work and points out the motivations that brought to the definition of the methodology described in chapter 3.

2.1 2.5D Perspective

The term $2\frac{1}{2}$D was firstly introduced into psychology by Marr [1982] as a representation of the human vision system. In computer graphics, a 2.5D scene manipulates a set of 2D surfaces in a 3D environment, simulating the relative depth between them. The purpose is to allow the users to watch a 2D character from any 3D angle. This problem is related to computer-assisted animation, which dates back to the early seventies [Burtnyk and Wein, 1971]. The connection is more evident with Catmull's work [1978], in which he underlines that the main problem of 2D computer-assisted animation is that a 2D character is a 2D projection of a 3D model which the animator has in mind. When a part of the character is covered, it cannot be reconstructed without that model; hence, information is lost.

The works in this section are classified into two different categories. The first one, in section 2.1.1, includes those works which describe and handle the layering problem. Layering is the simulation of the missing depth information for the represented 2D surfaces by assigning them a *layer*, which outlines how each one of them should behave with respect to other surfaces — be in front or behind them. The second category, in section 2.1.2, includes the works which allow watching a 2.5D cartoon character — a character composed by several different 2D surfaces — from any 3D view, generating a rendering of that cartoon from a new view.

2.1.1 Layering

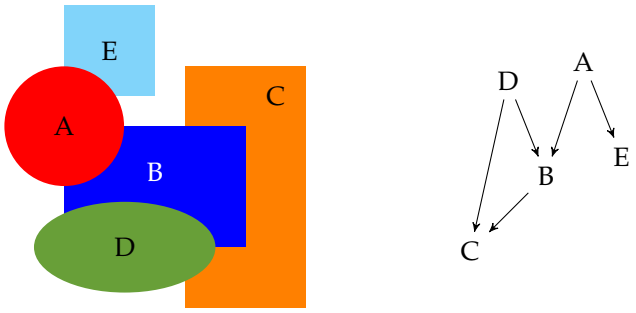


Figure 2.1: The classic approach to layering. On the left, the figure; on the right its DAG.

The concept of *layer* is mostly diffused in the drawing software domain. Users can draw sketches on different layers and place them over or under others in the same image. They can even change the order later. The classic representation of the layers involves the use of Direct Acyclic Graphs (DAG)

[Harary, 1969], as shown in figure 2.1. Research on this field tries to extend the concept of the layer, allowing interwoven surfaces — as in figure 2.2 — and more complex operations. Layers are an essential tool for 2.5D modelling. 2.5D scenes — and characters — are composed of several 2D surfaces together. Correctly layering them is fundamental to display them properly and thus to simulate the appropriate depth.

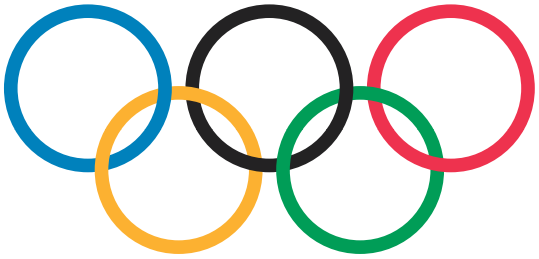


Figure 2.2: An example of a figure impossible to draw with the classic layer approach.

Figure 2.3: Some of the shapes realised with the drawing tool from [Wiley and Williams, 2006].

One of the first work to extend the more traditional way of layering is *Druid* [Wiley and Williams, 2006], a vector-graphics drawing system that allows the users to draw interwoven surfaces. While few older works supported the creation of knots, *Druid* allows creating more general shapes; it can draw any orientable two-manifold with a boundary. Figure 2.3 shows a few examples of shapes realised with *Druid*. To handle overlapping surfaces, previous drawing software represented the drawings as a set of layers, and each surface lays on just one layer. To determine the order of the layers, they used a DAG, which defined a partial order, thus preventing any interweaving. To allow their software to handle interwoven drawings, Wiley and Williams introduced a new representation for the drawing, which contained local information about the depth of sub-regions of the surfaces. While existing drawing software represented the shapes using their interior, *Druid* uses their boundaries and keeps the information of which curve stays above wherever it intersects another one. The interior of the shape is contemplated just in the final rendering step.

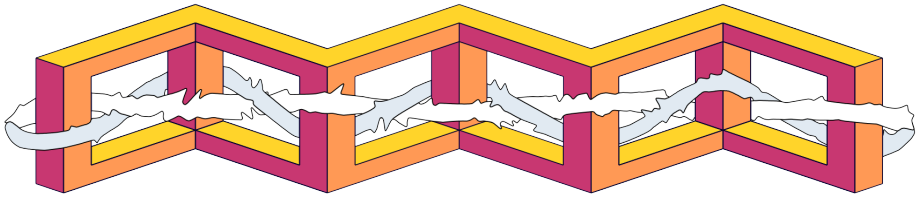


Figure 2.4: An example of what can offer the local layering, from [McCann and Pollard, 2009].

McCann and Pollard [2009] propose a more generalised layering approach. They extend the idea of *Druid* to obtain similar results with raster objects, not only solid fill colours. In their work, they handle the object ordering at a local level instead of global. To do so, they introduce a new data structure, the list-graph, to track the relative order of the objects for each region of overlap. Although their method does not support self-overlaps, it improved *Druid* on several technical aspects. Their labelling system is more efficient, as it is polynomial time, while *Druid*, in the worst case, involves an exponential time search. The rendering is faster in their system because the stacking order for the interior of the shape is known, while in *Druid* it has to be computed at rendering time. Moreover, McCann and Pollard offer proof that their editing operations are sufficient to reach all valid orderings. In the end, their representation of the shapes, which uses adjacent regions, is easily generalisable by giving a temporal extent to the regions. Figure 2.4 shows an example of a complex scene of overlapping objects obtained with this system.

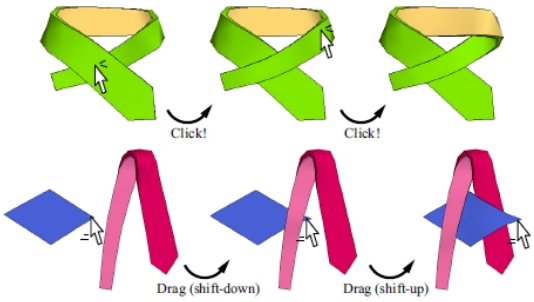


Figure 2.5: Local layering expanded to 3D objects as in [Igarashi and Mitani, 2010].

Igarashi and Mitani [2010] exploit the concept of layering and the improvements introduced by previous work — such as local layering from [McCann and Pollard, 2009] — to handle the editing of 3D objects with a 2.5D interface. Handling layering in 3D is more laborious than in 2D, as the system must consider interpenetration. Their method allows the users to swap two different layers and a layer-aware dragging system. A few examples are shown

in figure 2.5. Different from the local layering, this method uses 3D models; therefore, it supports layer operation for objects with self-occlusions and folds.

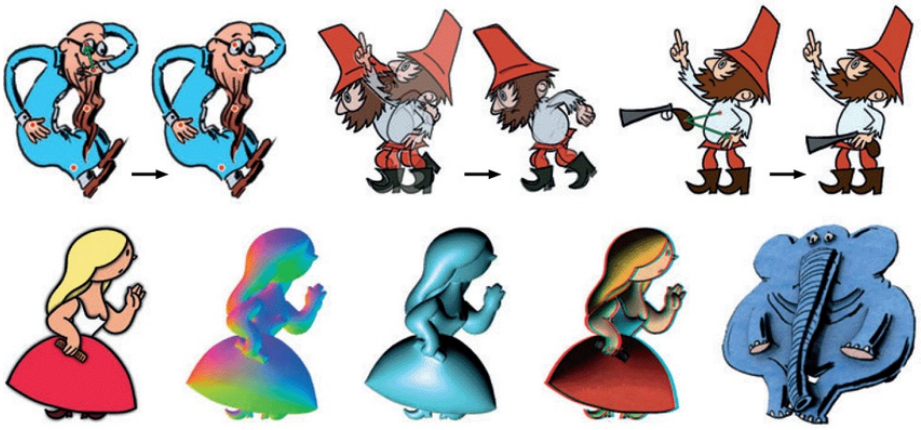


Figure 2.6: Some applications of the work from [Sýkora et al., 2010].

In the same year, Sýkora et al. [2010] present a method to recover the depth from a single image, particularly from hand-made drawings, providing a 2.5D pop-up matching the observer's perception of depth. Figure 2.6 shows some of the results obtain with this system. Their method operates on an input-provided set of sparse depth inequalities between different parts of the drawing. Using depth inequalities instead of absolute depth values is a significant advantage of this method, which tries to emulate the depth recovery mechanism of the human visual system. Depth inequalities are, in fact, more intuitive to assign compared with absolute depth values and faster to input. Their system computes the depth map by solving a quadratic problem. This method has two main limitations. The first one is the imprecise estimation of the contours, which leads to the production of several artifacts. The second is related to the assignment of depth inequalities. They are set using an oriented graph. If the system detects a cycle in the graph, it creates a new segment to avoid it. However, while it can automatically detect cycles, it cannot solve the conflict without interaction from the users.

Zhang et al. [2012] propose a semantically meaningful layer extraction method for cartoon animation. This approach is opposed to the local layering method from [McCann and Pollard, 2009]. Local layering approach is about composing different layers into a single image, while Zhang et al. focus on decomposing an image in several different layers. A key feature of the method is that the label is propagated automatically to all the frames after manually marking the desired layer in a keyframe. Afterwards, the method handles the missing regions due to the occlusion of overlapping layers. This approach

has several limitations. For instance, it can handle only animations with a fixed number of layer. Moreover, the extraction method often produces irregular boundaries, especially for the hair. The system cannot handle changes in occlusion order; hence, the layer order must be fixed over time. In the end, the animation has to be smooth between consecutive frames, although this is common for cartoon animation; therefore, this last limitation is less severe compared to the previous ones.

X. Liu et al. [2013] follow a path similar to [Sýkora et al., 2010] and exploit the layering to compute a pseudo depth map and thus produce a stereoscopic cel animation. To estimate the depth between the different regions of the image, they use the T-junction cue. As the T-junctions could not always be precise, they compute the ordering from several frames to reduce the noise and maintain temporal consistency. They also calculate a pseudo-depth map to make the depth of each region change smoothly over time. In the end, they compute two renderings of each frame from two different eye positions. This method is highly automatic. However, it cannot handle surfaces that gradually change their depth value; therefore, this method cannot compute an accurate depth, and the users have to solve this problem manually. Using a solution similar to [McCann and Pollard, 2009] to represent the layer at a local level could help to solve this problem.

Dalstein et al. [2014] introduce a new data structure, the VGC — Vector Graphics Complexes — to extend the range of objects that vector graphics software can draw. The most common representations of the layers in the vector graphic systems are stacked layers of paths and planar maps, which have significant limitations. The former cannot represent shared edges between two shapes, while the latter is unable to represent overlapping faces. VGC overcomes these limitations. It is designed to be a superset of multi-layer vector graphics, planar maps and stroke graphs, thus supporting several kinds of geometries incompatible with existing formats. Dalstein and colleagues also introduce topological operators to exploit the properties of the new data structure, such as glue and cut of vertices and edges.

2.1.2 2.5D Cartoon Modelling

This section analyses the works which generate a 2.5D model. These systems manipulate 2D layers to visualise the character — or object — correctly from any 3D direction.

Di Fiore et al. [2001] present a method to compute in-betweens automatically, exploiting a 2.5D model. Their method requires the users to manually provide a 3D skeleton of the character or specify the z-ordering for all the

extreme frames. To render a new frame from any 3D rotation, they compute the rotation angles relative to a 2.5D object with respect to a virtual camera. Then, they use these angles to search for a near extreme frame. The system linearly interpolates the positions of the control points while it takes the drawing order from the extreme frame. The necessity of manually providing the z-ordering for the 3D skeleton is a significant limitation of this method. The approach proposed with this thesis aims to use the computer vision technique of triangulation to automatically understand the 3D position of each part of the character.

Rivers et al. [2010] present a system to create a 2.5D cartoon model from 2D vector drawings. In contrast with Di Fiore and colleagues' method, they compute the 3D position of each component automatically by exploiting two drawings from two different views. They also propose a 2.5D interpolation instead of a simple 2D one, which consists of the interpolation of the 2D drawings while maintaining the 3D structure, thus simulating the rotation of the character. The method presented in this thesis follows a similar procedure, although it maintains all the character's components in the same 2D plane. This approach suffers from popping artifacts in the case of overlapping shapes. Moreover, the system cannot interpolate shapes when they are partially occluded. The method presented in this thesis handles the occlusions by using the triangulation and reconstructing an approximated 3D model of the character.

An et al. [2011] propose a method to generate a 2.5D model, like those from [Rivers et al., 2010], from a 3D model. The 3D model is segmented into different parts, from which then is extracted the silhouette from a set of viewpoints. The system uses these silhouettes as the 2D components of the 2.5D models. Although this system is capable of creating the 2.5D models correctly from 3D ones, it still requires a 3D model of the character, which is a significant limitation of the system. Depending on the level of detail of the character, modelling a 3D could result in more time consuming than draw the character from multiple viewpoints. Moreover, as mentioned in chapter 1, a 3D model does not take into consideration the distinctive traits of the 2D characters.

Furusawa et al. [2014] present a new method to simulate a quasi 3D rotation of a hand-drawn character from two keyframes, the front and the profile. In contrast with River's method [Rivers et al., 2010], they do not move the character's parts in three dimensions, but they keep them on the same plane. The method is based on two operations. The first one is the computation of the component's position, which relies on the centroid of the shape. The second one is the interpolation of the component's shape. A significant limitation

of this method is the "quasi 3D" rotation of the character. The rotation is limited to the horizontal space. The method introduced in this thesis, by contrast, gives more freedom to the users, allowing them to watch the character from any direction.

Kitamura et al. [2015] attempt to enhance River's approach [Rivers et al., 2010] by creating a 2.5D model from two images. In contrast with An's method [An et al., 2011], which used a 3D model to create the 2.5D one, Kitamura et al. use images, which are more accessible and less time-consuming to produce. They segment the two pictures and find the region correspondences between them by using the similarity. To estimate the 3D position of each component, they use the same as River, the 2.5D interpolation. The main limitation of this method is its inability to handle occluded parts. The users have to complete manually any region which is partially occluded in the input images.

C.-K. Yeh et al. [2015] propose a method to generate simple hair animation for cartoon characters. This method employs a 2.5D model for cartoon hair generated from a single view. The first step of the method is the segmentation of the input image. After segmenting each hair strand, the authors introduce a new layering method based on the *amodal completion law* of Gestalt psychology. Next, they propose a method to fill the occluding parts of the hair strands automatically. As this method is for cartoon characters, it does not need to fully reconstruct the hair, but just the 2.5D visible part that needs to be animated in the 2.5D model. Finally, they generate the skeleton for animation and manipulation. The layering method proposed by this work does not consider local layering [McCann and Pollard, 2009], thus preventing the method to work with intertwined hair strands. Moreover, the method does not work with messy hairstyles, as the segmentation would be impossible to generate.

Entem et al. [2019] propose a system to automatically extract the structure and the layers from drawings and to manipulate the elements of the character to generate new poses. Their method consists of decomposing the input drawing into different parts, based on connected internal silhouettes and inner closed contours. These contours offer information about the layers and overlapping parts. To compute the structure of the drawing, they introduce the "radial variation metric", a 2D variation of the *volumetric shape image* [R. Liu et al., 2009] used for the segmentation of 3D models. They also introduce a recursive algorithm to identify parts in more complex sketches and assign them the correct depth-layer. During the decomposition, the system records additional information to put the extracted parts together again in the new poses, such as where the contours should be erased, in order to create new poses. The generated output uses the VGC format [Dalstein et al., 2014] discussed

previously. One of the main limitations of the method is that it cannot handle self-overlapping parts.

2.1.3 Summary

So far, the most significant works about layering have been analysed. Some of them, such as [Wiley and Williams, 2006; McCann and Pollard, 2009; Dalstein et al., 2014], extend the classical concept layers in drawing software to allow the users to draw interwoven surfaces. Others, as [Sýkora et al., 2010; X. Liu et al., 2013], exploit the layering to compute an "approximate" depth map in cartoon characters for different purposes. The work of [Igarashi and Mitani, 2010] shows an unusual application of the layering in a 3D context. In the end, [Zhang et al., 2012] demonstrate how to extract semantically meaningful layers from cartoon animations.

Next, an overview of the most notable methods to generate a 2.5D model of a character have been presented. While the main focus of the 2.5D modelling is the automatic generation of inbetweenings from two views of cartoon characters [Di Fiore et al., 2001; Rivers et al., 2010; An et al., 2011; Furusawa et al., 2014; Kitamura et al., 2015], the work of [C.-K. Yeh et al., 2015] and [Entem et al., 2019] demonstrate that it can be employed for other purposes, too, such as the animation of single-view 2D characters.

The method for 2.5D modelling presented in this thesis is more close to [Di Fiore et al., 2001; Rivers et al., 2010; An et al., 2011]. Similarly to these works, its objective is to create a full 2.5D model to be used in a 3D environment. However, the novelty of the method presented in this thesis is in the input. In fact, it works with existing 2D cartoon characters, while these previous methods require drawing the character from scratch with their tools.

2.2 3D Modelling from 2D Sketches

The automatic generation of 3D models from 2D sketches poses a difficult challenge. A 2D sketch represents just a silhouette or a contour of a character, and the insufficiency of information may lead the computer to misinterpret it.

Several works offer a complete and detailed survey of this topic [Olsen et al., 2009; Cook and Agah, 2009; Ding and Liu, 2016]. This thesis briefly analyses the principal works and categorise them per approach. Five different categories are typically recognised.

2.2.1 Early Works

Zeleznik et al. [1996] introduce one of the first approaches to using sketches to generate 3D shapes. The idea is to employ the sketch's geometry to define both the 3D shape to be generated and its details. For instance, the input sketch of a circle would be used to generate a sphere with the same radius. As this method is only a first attempt to model 3D shapes from 2D sketches, only a small set of basic shapes are supported. Other limitations of the method concern the usability of the method and its interface.

Igarashi et al. [1999] extend the idea from [Zeleznik et al., 1996], but their system allows the generation of 3D models from freeform sketches. Users do not need to learn specific gestures to generate a set of shape but can freely draw any shape. First, the system generates a polygonal mesh with a constrained Delauney triangulation and refines it. It uses, then, the chordal axis [Prasad, 1997] to decide which points to "inflate", of an amount based on the distance from the contour. The system also allows the extrusion of new parts, cut them and smooth the polygons.

Other works tried to extend the number of shapes that [Zeleznik et al., 1996] could generate. Recognising more shapes made it more difficult for the system to recognise the correct shape to generate accurately; therefore, the systems provided interactive suggestions.

One of these suggestion systems is introduced by Pereira et al. [2000] in their software, GIDeS, implemented as an "expectation list". Whenever the users draw a sketch, the system shows a window with the most plausible interpretations of the stroke. The users can then select the desired shape. This approach helps to avoid misinterpretations of the sketch in case of ambiguities but makes the whole process of modelling slower, as it needs a selection of the correct shape for each stroke.

Igarashi and Hughes [2001], instead, propose a "suggestive interface". Similarly to the expectation list, it displays suggestions to the users on the interpretation of the input strokes. However, in this case, users can select an explicit selection of elements in the scene to get suggestions regarding those elements. Additionally, they can input multiple sketches and then pick a suggestion from the system, accelerating the whole process.

2.2.2 Inflation Approach

The inflation of the 3D model is one of the first approaches that researchers exploited for sketch-based modelling. The inflation methods generate the 3D surface by *inflating* the points of a 2D surface, meaning that they first generate the 2D mesh and then compute the z value of the vertices. One of the first

works to employ this technique is [Igarashi et al., 1999], discussed in the previous subsection. Several methods in this category took inspiration from this work.

Karpenko and Hughes [2006] improve on [Igarashi et al., 1999] by presenting a software, *SmoothSketch*, which can take as input sketches with hidden contours, T-junctions and cusps. Their method, based on [Williams and Hanson, 1996], consists of three steps. The first is the inference of the hidden contours, which, differently from [Williams and Hanson, 1996], includes T-junctions and cusps. Secondly, they generate a topological embedding from the completed contour and map it to $\mathbb{R}^2$. Finally, they merge in a single mesh the individual panels of the mapping in $\mathbb{R}^2$ and inflate it to a smooth embedding in $\mathbb{R}^3$.

Nealen et al. [2007] propose a new method to handle the sketches. In their software, FiberMesh, each stroke guides the inflation as in [Igarashi et al., 1999], but they are kept visible on the generated model as "control curve". The users can make operations on these curves to edit the 3D shape. They can also add new strokes, which will be used to update the shape by solving an optimisation problem.

Olsen et al. [2011] introduce NaturaSketch, which is a middle ground between [Igarashi et al., 1999] and other gesture-based methods. As the software allows the users to generate a 3D model from a free-form 2D sketch, it also introduces several gestures to apply transformations to the generated model, such as extrusion, bumps or holes. The system employs the distance transform to compute the minimum distance of each pixel in the image from the contour to generate the inflation. Then, any vertex of the planar mesh generated from the image is inflated with a circular mapping function of the distance to get a more smooth result.

2.2.3 Direct Approach

The methods described in this subsection generates the 3D surface directly, without the intermediate step of the 2D surface.

Lee et al. [2008] propose a sketching framework to generate 3D models of architectural structures. The process of their framework can be split into two distinct parts. The first is the generation of polygons from connected strokes in the drawing, which employs a heuristic method. The second part is the inference of the position relationships between the different generated polygons by solving an optimisation problem. The authors considered common architectural properties in the energy function, such as aligned and parallel axis and symmetric surfaces.

B. Xu et al. [2014] introduce a sketch-based modelling system focused on design sketches. They introduce a mathematical framework based on insights from perception and design literature. They noticed two key facts about the designers' drawings: first, they favour viewpoints to maximise the reveal of 3D shape information. Second, they draw curves to highlight properties of the shape, such as curvature, symmetry and parallelism. Their algorithm detects and enforces these properties and accounts for their impact on the generated 3D curves. Moreover, a regularisation process balances between the enforcement of these properties and the fidelity to the sketches.

Entem et al. [2015] propose a method to generate a 3D model from side-view sketches of animals, assuming they are symmetric. The first step of the method is the refinement of the input sketch, which the system converts into a parametric curve in order to identify the strokes which belong to the same part and complete the contour if it is open. The second step is the structuring and layering of the different parts, similar to the layering performed for the 2.5D modelling, which has been described in section 2.1.1. The final step is the generation of the 3D surface, which exploits the medial axis. They first compute, refine and simplify the medial axis for each part of the character. Then, they compute the 3D surface using [Zanni et al., 2013], by exploiting the skeleton and its radius function. The models of the different parts are then correctly placed following the structuring computed in the previous step and merged with a blending operator.

You et al. [2020] introduce a method to generate a 3D model using Partial Differential Equation (PDE) surfaces — defined as sculpting force-driven shape representations of interpolation surfaces. The system consists of global modelling and local modelling. The global modelling consists of creating the whole character model from three levels of sketched profiles in three orthographic views and deforming a template model to match the sketches. The local modelling is used to add details to the created character model, using four shape manipulation techniques: PDE-based, generalised elliptic curve-driven, sketch assisted and template-based shape manipulation. The final model is represented as a collection of surfaces modelled with generalised elliptic curves and PDE-based surfaces.

2.2.4 Retrieval Approach

Instead of directly generating the 3D model from the sketch, the works in this category exploit a database and use the input sketch to find a close match in it.

X. Chen et al. [2008] propose a system to generate a realistic-looking model

from freehand sketches, with a focus on architectural designs. While the users draw lines, the system identifies junctions, edges and faces. Then, it interprets the 2.5D geometry and analyses the extracted topology. Users can also include information about the geometry and the textures through sketches. The system then matches the partial sketches with detailed models in a database. The final result is a realistic 2.5D model of the building, complete with textures.

Gingold et al. [2009] present a system to generate 3D models from freehand sketches. The system is based on two kinds of annotation that the users can provide on top of the sketch. First, the users place primitives, based on common sketching conventions, from a single view of the model. Second, they provide annotations of higher-level semantic information, such as alignment, symmetry or connection curves. With these two sets of annotation, the system can generate consistent 3D models even for inconsistent 2D sketches.

Eitz et al. [2012] introduce a new method for 3D object retrieval from sketched feature lines. Their study observes that input sketches have a large local and global deviation from the shape the users are trying to draw. Besides, the sketches provide only partial information about the projection of the shape. Their system addresses these problems, and it is based on the following desirable characteristics. First, the partial matching of the feature lines. Second, the analysis of all the potential view directions. Third, the accounting of global and local deformations. The approach they follow is based on the visual analysis of the meshes. They sample all the likely view directions of the mesh, and for all of them, they generate line drawings with different rendering techniques. Then, they encode all the drawings with the bag-of-features approach [Mikolajczyk et al., 2005; Sivic et al., 2005].

Shtof et al. [2013] propose an approach to 3D modelling from 2D sketches based on the snapping of 3D primitives over the input drawing. The users provide a semantic classification of the strokes of the input sketch by identifying the feature curves and the contour curves. Then, the users drag a 3D primitive over the sketch, and the system will automatically snap it in real-time in the correct location and with the proper orientation. If the users already placed other primitives previously, the system will automatically infer the geosemantic constraints to compute the final snapping.

K. Xu et al. [2013] introduce a new framework to generate entire 3D scenes from a collection of freehand sketches. The generated scenes are semantically valid, as the framework bases the retrieval of the 3D models on the recognition of the objects as a group. Differently from other methods, this approach retrieves and places the 3D objects in the scene by processing and analysing the objects together. This approach reduces the amount of information required from the users. The identification of the context in an example scene

repository helps in the identification and retrieval of the single objects as well.

2.2.5 Shape Editing Approach

The methods described in this subsection do not generate a 3D model directly but use the sketches to edit an existing model.

Nealen et al. [2005] present an interactive system to apply deformations to meshes with sketches. The users draw a sketch or select and crop a silhouette to define the region to deform. The system uses this region as a reference curve. Then, the users draw the target curve. The system computes the correspondence between the two curves. This method uses the discrete Laplace and Poisson models in order to preserve the local and global geometry.

Zimmermann et al. [2007] propose an improvement of what presented by [Nealen et al., 2005] for what concerns the reference curve. The main aim of the work is, in fact, the automatic inference of the reference curve and the region to deform, thus allowing the users to input just the target.

Kraevoy et al. [2009] aim to produce new models by deforming a 3D template with contour drawings. Their system solves an optimisation problem to compute the correspondences between the drawn contour's points and the model's vertices, which is then aligned and deformed to match the sketch.

2.2.6 Summary

In this section, the main works regarding 3D modelling through 2D sketches have been reviewed. Five different categories have been defined.

- The early works are based on the understanding of specific gestures which generate a corresponding 3D model. These methods are limited by the number of gestures that the author implemented in their software.
- The methods based on inflation can generate 3D models from any 2D drawing, as they compute the 3D surface from the 2D shape defined by the sketch.
- With the direct approach, the systems generate a 3D model by directly interpreting the 2D sketch without inflating it. These methods usually solve an optimisation problem to compute the position of the 3D vertices.
- The methods that make use of a database to generate a 3D model or scene are categorised in the retrieval approaches.

- The shape editing category, in the end, includes all those methods that, instead of generating a 3D shape from scratch, use the 2D sketches to change existing models.

For the objective of the thesis, as it aims to generate 3D models for cartoon characters, any method that retrieves existing elements from collections of any kind cannot be employed. The same applies to shape editing methods, as the input of the system will be just a 2D cartoon character.

This thesis proposes two different methods. The first one, based on inflation, is similar in many aspects to the works in that category. The novelty of the method presented in this thesis is the combination of the inflation technique with a surface registration method. A typical inflated model, in fact, does not keep any detail of the 2D character but inflates it uniformly along the whole surface. Instead, combining the inflation with the registration allows to interpolate between two models inflated from two adjacent perspectives, thus obtaining those details while the camera rotates around the character.

For what concerns the second method, it belongs in the direct approach category. The main novelty of the approach consists of using multiple drawings of the character from different perspectives. These typically available in animation production in the form of turnaround and allows the system to use this data to include features in the model that are visible only from some of the points of view. While there are other works that generate a 3D model from multiple sketches from different points of view, these usually require specific sketches for their tools, with numerous additional sketches with additional information. The method proposed in this thesis, instead, works with existing images of 2D cartoon characters and requires only a few correspondence points from the user.

2.3 3D Animation from 2D Sketches

Analogously to the generation of 3D models from 2D sketches — section 2.2 — posing a 3D character using a 2D input presents a difficult challenge.

The main problem with this method is that the sketches, in their 2D nature, do not possess any depth information; therefore, associating a stroke with a 3D component of the character could be difficult. The system proposed with this thesis faces a similar problem. The system generates a 3D animation from the input 2D one. The same ideas from the works in this section can be used to analyse the 2D animation and correctly represent it in the 3D space.

There are three main approaches to this problem. The first one uses a database and retrieves the pose by comparing the sketch with the records

in the collection. The second one matches the sketch with elements in the character's model and directly poses that character. The last approach uses the sketch to edit the character's pose in time.

Before analysing the works in these categories, however, this section introduces a few works worth mentioning because they are the first approaches to the character posing through sketches.

2.3.1 Early Works

These works are the first ones that employ sketches to pose characters. They do not pose a proper model, but they create a representation of the character's skeleton hierarchy.

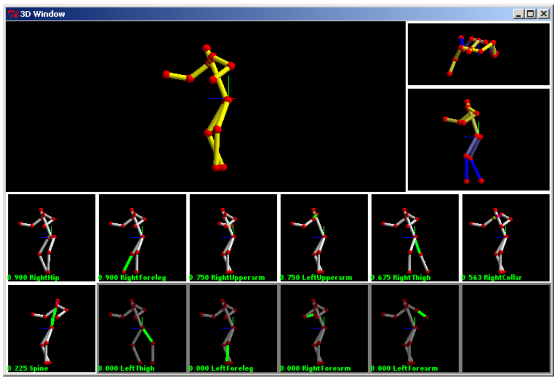


Figure 2.7: The pose selector from [Davis et al., 2003].

Davis et al. [2003] are the first to introduce a specific method to pose articulated figures with sketches. Previous methods provided tools just for precise 3D positioning, and they were not suited for posing articulated characters. Davis and colleagues' tool requires as input a keyframe sequence sketched from the artist and a manual annotation of those with the bone structure. Mostly, the users have to draw the joint position with circular dots and the connectivity with thin lines connecting them. Then, the system automatically labels the stick figure. The system requires a template of the skeleton, and the users have to specify both connectivity and bone length manually. Once the users submit the skeleton template and the labelled stick figure, the system will reconstruct all the possible 3D poses that match the input sketch and that have a valid ranking score, which is computed according to a set of heuristics. Their ranking score orders the 3D poses, and the users can select the pose that is the closest to his idea. The interface is shown in figure 2.7. While taking into account that Davis and colleagues are pioneers in sketch-based character posing, their tool has several problems. First, the necessity of the users to spec-

ify the template of the skeleton through a configuration file and not directly through the application could be too complicated for average users of the tool. Moreover, even if the manual annotation of the sketch is not a complicated or long task, if the users have a significant number of keyframes, annotate them all could be a laborious work that the application could automatically do.

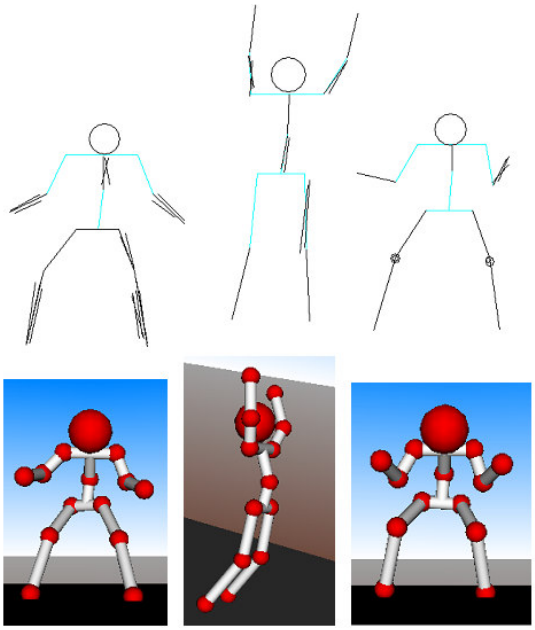


Figure 2.8: Sketches and the resulting posing obtained with [Mao et al., 2005]. As can be seen from this picture, the thickness of the lines is used to determine the depth of the body parts.

Mao et al. [2005] introduce a method that is a direct evolution of [Davis et al., 2003]. They take into account all the problems and try to solve them (not with a definitive solution, however) while adding new features. With their tool, the users do not need anymore to specify the template of the skeleton manually; in fact, now they can select one from a list categorised by gender, ethnicity and age. Then the users draw the stick figure by using the thickness of a connection between two joints to express the depth of that body part , as can be seen in figure 2.8. However, the drawing process is guided, and the users cannot draw freely, which implies that there cannot be body extensions. Finally, the system recognises the skeleton and reconstructs the 3D character with a heuristic system similar to the Davis' one. Although this method is a vast improvement from the previous attempt, there are some limitations. While it is simpler for the users to select the template than manually configure it, it is also true that the tool is limited, since they cannot define custom templates. Regarding the guided drawing, it indeed is a suitable approach

for apprentices or for people who cannot draw, but for experienced users, this could result restrictive.

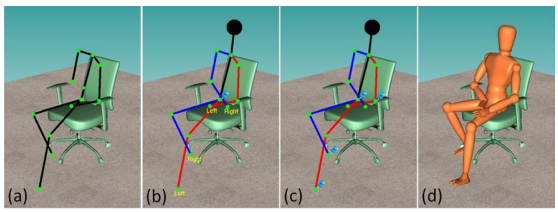


Figure 2.9: An example of a sitting character posed with [Lin et al., 2012].

While Mao et al. [2005] focuses on improving [Davis et al., 2003], Lin et al. [2012] attempt to solve a new problem: the interaction of the character with the environment. In particular, it focuses on sitting characters. In figure 2.9 there is an example of the sitting posing. They use a sketching interface similar to [Davis et al., 2003], but with some modifications. The users cannot freely draw the sketches, as the application drives the drawing. However, they can choose the first joint to draw and start sketching from there. The users here cannot select or define a template for the character, which is fixed. While drawing, Lin's tool allows the users to set some joints as *pinned*; these are the joints that are in contact with a surface, so, during the model reconstruction, the model's areas that touch the surface will be adequately handled. The 3D character reconstruction is obtained by solving an optimisation problem. This tool focuses on the interaction between the character and the environment, and the authors do not attempt to solve the problems that affected previous works [Mao et al., 2005], such as the driven drawing and the predefined templates. Apart from that, the limitations regard mostly the poses that the sitting character can assume. Despite these constraints, so far, this is the only work that uses sketches to pose the character while handling its interaction with the environment.

2.3.2 Retrieval Approach

These works are based on a database to retrieve the character's pose from the sketch. They compare the drawing with the records in the database and return the most appropriate pose.

In 2011, Wei and Chai [2011] stood utterly apart from the previous works. They exploited a technique to extract prior information from an extensive mocap database and used it to interactively pose human characters. In their tool, Wei and Choi offer two different interfaces: the direct manipulation of the joints and the sketch-based interface. The first one allows the users to

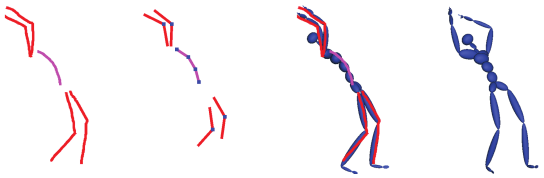


Figure 2.10: The sketching interface of [Wei and Chai, 2011].

edit the pose in real-time in a variety of ways, such as dragging any joint or setting the distance between two joints. Instead, the sketching interface , shown in figure 2.10, allows the selection of any limb or the torso and to draw the desired location. The 3D pose is then obtained by solving an optimisation problem which allows retrieving from the database a vector with the joint-angle space information. Compared to the previous works, this method has the advantage to always generate natural poses on the ground that the system retrieves poses from a motion capture database, even if sometimes it could return a different pose. However, the sketching interface is minimal, and the necessity to select the body part for each sketch that the users are drawing slows down significantly the process.

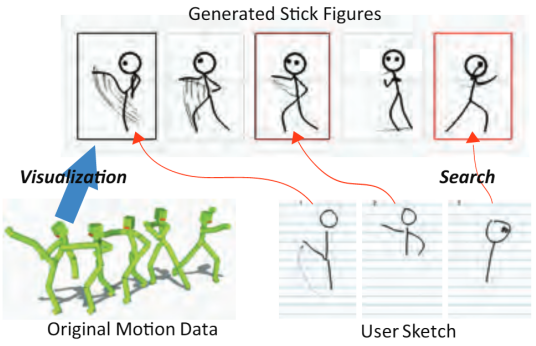


Figure 2.11: The visualisation of the 3D character as stick figures in [M. G. Choi et al., 2012].

M. G. Choi et al. [2012] share the same idea as [Wei and Chai, 2011] to exploit a database for the animation. However, while Wei and Chai [2011] retrieves just a keyframe, Choi's system fetches an entire motion from a single sketch, in particular from a stick figure. Moreover, the system visualises the motion as 2D stick figure images, as shown in figure 2.11. The tool includes an algorithm to convert the motion data into a sequence of stick figures. In this way, it retrieves the motion by comparing the sketch of the users with the set of stick figures obtained from the motion data. The tool does not compare the image to recognise the sketch, but it recognises the human shape inside the

sketch and compare the chosen features only. The system returns a result from the first stroke and updates it as the users add other strokes. Although this work is not a system to pose the character with sketches, the way it operates and the operation that it does to return the results could be compared to the methods described so far. Although the system does not allow such a thing, it should be possible to retrieve the 3D character pose from the motion data at the moment from which the stick figure was extracted.

2.3.3 Shape Editing Approach

This method exploits sketches to manipulate the characters' model directly. While there are similarities with the works in section 2.2.5, the methods described in this section operate on the skeleton of the character, and the 3D model changes as a consequence of a skinning operation previously applied.

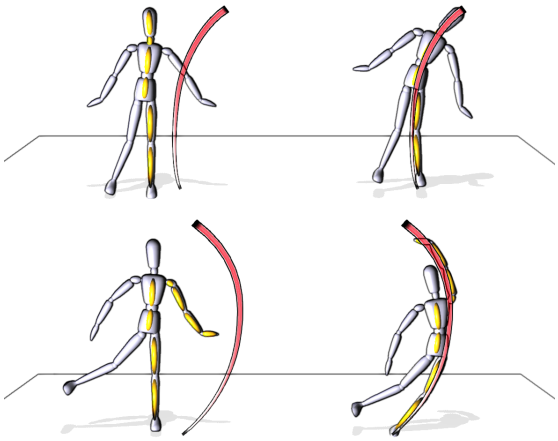


Figure 2.12: Two examples of the posing using [Guay et al., 2013]. The yellow line represents the body line.

A significant change happened with Guay et al. [2013]. They take the concept of the line of action, a technique used in drawing to grant a dynamic look to a character, and, while giving it a formal definition, they use it to pose a 3D character with a single stroke. Figure 2.12 shows an example of the line of action. Three problems must be solved to pose the character: the correspondence between the stroke and the body components, the posing itself and the selection of the body line, which is the selection of the set of bones that the line of action will modify. The first two problems are solved with two different optimisation problems. For the body lines, instead, Guay allows two different solutions: a manual selection of the bones or an automatic one that selects the line with the smallest energy in the minimisation of the posing. In order to obtain more complex poses, users can draw multiple lines. The line

of action is a significant improvement of the character posing using sketches. This tool allows the users to pose the character in a straightforward and fast way. However, this is the most significant limitation of the tool. It does not allow more experienced users to use a semi-automatic approach that would allow them to obtain results, probably in a faster way, bounding them with the fully automatic procedure.

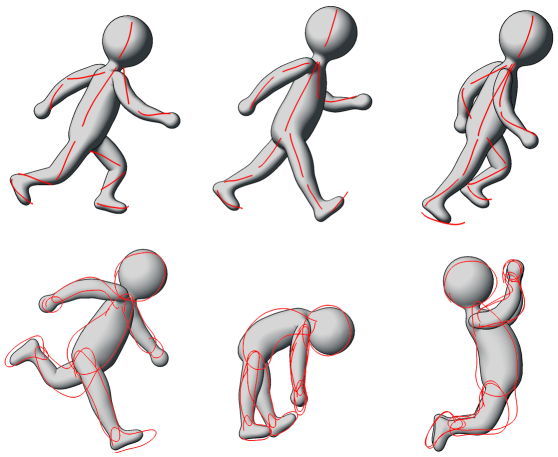


Figure 2.13: A few poses obtained with [Hahn et al., 2015]. The red lines are the sketched curves.

In 2015, Hahn et al. [2015] introduce an original concept: the sketch abstraction. Fundamentally, it is a way to connect the character's rigging to the 2D input sketch. It can be provided with the model by creating and rigging a set of curves, or it can be created at runtime by drawing a curve on the top of the character. The abstraction is bound to some points in the mesh. It then establishes a correspondence between the abstraction and the input sketch, which can be computed with a straightforward in-order correspondence by considering the drawing direction or using the closest-point matching between the two sets of points. Therefore, the final posing is retrieved by solving an optimisation problem. Few results can be seen in figure 2.13. The sketch abstractions are an interesting technique, which can expand and has applications even beyond the character posing — for example, it could be used to create a small database of body parts and design new simple models by using sketches. Nonetheless, it has substantial limitations, the greatest of which is intrinsic in its purpose. In fact, as support for posing, it requires the artist to learn a new paradigm. Moreover, for every keyframe, they have first to draw the abstraction and only then the pose. Although this system produces good results, it is also true that previous works as the line of action allow the artists to get good results with already known concepts.

2.3.4 Time-based Approach

Finally, this approach gets over the keyframing to manipulate the pose of the character in time.

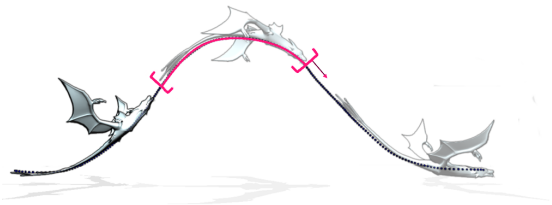


Figure 2.14: The space-time curve that generate a dynamic line of action from [Guay et al., 2015].

In the same period as [Hahn et al., 2015], Guay et al. [2015] present a new work with an entirely different way to pose characters by using sketches. They introduce a sketching concept distinct from the keyframing, which has been the approach for all the analysed works so far. The space-time sketching concept allows the animator to draw a fully coordinated movement, including the deformation of the shape over time, with a single stroke, which is called the *space-time curve*. The keyframing could still be used with this approach by drawing multiple lines of action , and it is shown in figure 2.14. From this curve, the system initialises a dynamic line of action which is associated with a specific body line in the character model. The space-time curve allows the users to create different kinds of motion, such as a path-following one, in which the character moves through the curve, a bouncing one and a rolling one. Then the users can refine the animation by adding several keyframes at different times in the space-time curve, even from multiple viewpoints, by adding periodic wave movement to the dynamic line of action or by specifying twists along the space-time curve. This method has the significant advantage of creating entire motions directly with a single stroke and refining them straightforwardly. However, it has several limitations too; the greatest one is that this method works fine with a limited category of creatures, but not with humanoids or, in general, with characters with two or more legs.

The space-time sketching concept has been developed and evolved by B. Choi et al. [2016] in their work, SketchiMo. Such as [M. G. Choi et al., 2012], his tool is not a tool to pose characters, but it allows for editing motions of articulated characters through only sketches. Choi introduces two new constructs, the sketch targets, which are the regions of the body that the users want to modify, and the sketch space, which is how the sketch target will be shown to the users. Two examples of sketch space are illustrated in

Figure 2.15: Two different examples of sketch space from [B. Choi et al., 2016].

figure 2.15. After the selection of the sketch target, the editing time and the camera view, the users can draw the new motion. The tool will fit it in the defined slice of time, and the result is immediately shown to the users. This tool gives the users the chance to edit in a natural way complex animations. However, its most significant limitation is that it cannot create an animation from scratch.

2.3.5 Summary

This section presented a brief analysis of the most important works about sketch-based character posing. This research topic evolved relatively quickly. However, each of the described approaches has unsolved problems.

The main issue of the database approach is that in case the sketched pose is missing in the database, it will be impossible to obtain that pose for the character. Moreover, a large database will require a significant amount of storage space available. For what concerns the shape editing approach, the biggest problem is that sometimes just one sketch is not enough to obtain the final pose desired, but the users have to draw multiple sketches, sometimes from different perspectives. Finally, for the time-based approach, the main problem is the difficulty to pose characters directly with these methods; they are more suited to edit already existing animations.

The method presented in this thesis belongs to the shape editing approach. It introduces two main novelties, compared to previous work. First, it uses the same concept of *sketch abstraction*, introduced by Hahn et al. [2015], but introduces a way to compute it automatically, whereas in the original work it was necessary for the user to manually sketch it. The second novelty is the method to solve the optimisation problem. In fact, the method presented in this thesis employs two techniques, the *softassign* and the *deterministic annealing*, together

with a *Robust Point Matching* algorithm, which is less inclined to find local minima and betters handle outliers.

2.4 Sketch-based methods and deep learning

In the latest years, deep learning methods have risen in popularity in almost every sector of Computer Science, and of course, Computer Graphics and Computer Vision are no exception. This section analyses some of the works based sketch inputs that employ deep learning, in particular for what concerns modelling and posing.

Han et al. [2017] introduce a method focused on the generation of 3D faces and caricatures from 2D sketches. This method allows the user to draw freehand sketches that represent the contour of facial features. The authors introduce a deep regression network based on a Convolutional Neural Network (CNN) to infer the 3D face from the 2D sketch. They also publicly released a new database they built for training and testing with numerous identities, expressions and levels of exaggeration.

Delanoy et al. [2018] propose a system to generate the 3D model with multiple sketches and an iterative approach. The CNN that they employ predicts the occupancy of the lines of the sketch in a voxel grid, but they complement the network with an updater CNN, that refines the result with new drawings from different perspectives. Both CNNs are trained with a collection of generated contours from hand-modelled shapes and procedurally generated abstract shapes.

Li et al. [2018], instead, use the CNN to infer the depth and normal maps of the surface from the 2D sketch. Moreover, they use an intermediate CNN to deduce the flow field of the surface — used to guide the surface reconstruction to have more regular surfaces — and produce a confidence map — which measures the ambiguity for the data fitting. The user can also provide optional depth values and strokes for curvature hints to reduce ambiguity. Their CNN is trained with a dataset of numerous 3D models rendered with a non-photorealistic line rendering.

Akman et al. [2020] propose a method that generates 3D models of humans and their pose from a 2D stick figure. The system employs a Variational Autoencoder network, trained with pairs of 3D — obtained by the SCAPE 3D human figure database — and 2D points — obtained by projecting the 3D points on a 2D surface. Their neural network ties together the 2D and 3D representation and allows to generate a 3D point cloud from the 2D stick figure. Their system can also generate a 3D animation through interpolation.

2.4.1 Summary

We have presented some of the works about sketch-based modelling and posing that involve deep learning. However, the methods presented in this thesis do not employ this technique, despite the advantages it provides. In fact, this thesis focuses on the generation of 3D models and animation for cartoon characters. Cartoons are, clearly, very different from real persons or animals, and they can represent creatures limited only by the imagination of the artists, and that might be very different from those in any training set. Additionally, with a machine learning approach might be challenging to obtain a specific style every time, so it would be necessary to adjust the algorithm every time to obtain the desired result. For this reason, this thesis employs more traditional optimisation methods to represent these kinds of characters in 3D.

2.5 Animation Pipeline

This section describes and analyses a typical pipeline for animation production, describing the differences between the production of 2D and 3D and for the production of video game animation. Of course, there is not a *standard* pipeline for animation, as each production is different and has to adapt the pipeline to specific requirements. However, the main structure is similar, and it is described in this section.

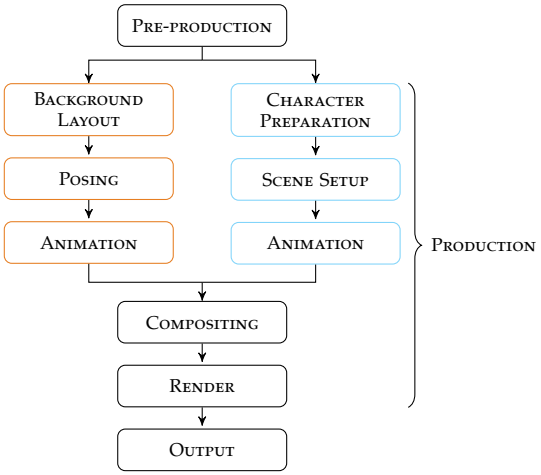


Figure 2.16: The pipeline for 2D animation. The steps are coloured based on the following convention:
Traditional Pipeline Cutout Pipeline

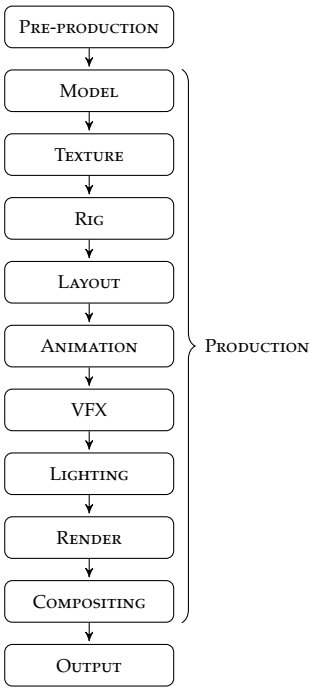


Figure 2.17: The pipeline for 3D animation.

2.5.1 Animation

Pre-production

The pre-production stages are very similar between 2D and 3D. Usually, these stages consist of the research, design and planning for the entire project. The writing of the story, the storyboard, and the design of characters and environment take place during the pre-production stage.

Production

The production stages include all those stages that contribute to the generation of the assets and all the elements included in any scene. This is where the differences between 2D and 3D start to emerge.

For what concerns the 2D animation, there are two different branches of the pipeline: traditional and cutout. The *traditional* pipeline is the original method, where every frame of the animation is drawn manually — historically on paper or cel, but today there is numerous software that supports traditional animation. In this branch of 2D animation, the first stage of production is the **background layout**. In this stage, the artist uses the storyboard to prepare the background, which can be drawn or assembled. This stage is followed by the **posing**, where the artist draws the characters in the main pose taken

from the storyboard. The next stage is the **animation**. Depending on the size of the studio, this stage might be slightly different. Bigger studios usually let the animators draw the key poses, while a specific department handles the inbetweens, but smaller studios let the animators draw both.

In contrast, the *cutout* pipeline does not let the animators draw the animation frame by frame, but instead the characters are split into several body parts, and the animation is carried out by moving these pieces each frame. This is also how stop motion animation work, and it is also very similar to 3D animation. In this branch of the pipeline, the first stage is the **character breakdown and rigging**. This stage includes the division of the characters into multiple body parts and their rigging, which consists of preparing the hierarchy of the body parts and set the appropriate pivot points. The next stage is the **scene setup**, similar to the layout for the traditional animation, where the background and the characters are placed in the scene. Next is the **animation** stage. With this type of animation, it is common that the animators draw the keyframe only, while the animation software interpolates the inbetweens. In some animation production, the animation stage might be mixed between traditional and cutout, where although the characters are split in body parts and rigged, some frames are manually drawn.

For both branches, the pipeline continues with the **compositing**. This stage consists on assemble all the elements together. This includes background, animation and sounds. In this stage are also created the camera moves and other necessary motions — for instance, for the background. Additionally, in this stages are added effects, including shadows and highlights. When the compositing is done, the last stage is the **rendering**, where the movie file is produced. This pipeline is shown in the diagram in figure 2.16.

The pipeline for the 3D animation is similar to the cutout 2D animation, but with few differences. The first stage of the 3D production is the **modelling**. This stage is the production of the 3D geometry for every element of the scene. The modelling is followed by the **texturing**, which is the process of creating an image that will be applied to the model to provide colours and surface properties. During this stage, the artists also create a UV map, which is needed to project the 2D texture correctly on the 3D surface of the mesh. The next stage is the **rigging**. Differently from the rigging of the 2D cutout pipeline, in 3D, the artists create a bone structure, which is then bound to the model with a process called skinning, which assigns a weight to each vertex of the model for each joint of the skeleton. Usually are rigged only the models which will be animated. Once the rigging is complete, the **layout** stage is where the artists place the elements in the scene. The following stage is the **animation**. Similarly to the animation in the 2D cutout pipeline, the anima-

tors in 3D usually animate only the keyframes, and the animation software interpolates the rest. Animation is followed by the **VFX**, where several visual effects are applied to the scene. These effects are complex to be manually animated; therefore, the software will simulate them. Then, the **lighting** is applied to the scene. The lighting needs to be baked to the scene, and depending on its size, it could take some time. In 3D, the scene is **rendered** into multiple layers, which include the objects, the background, the foreground, the shadows, etc. In the **compositing** stage, the artist put together all the rendered layers to produce the final output. The 3D pipeline is shown in figure 2.17.

2.5.2 Video Games

While both animation and video game production does not have a standardised pipeline, the animation one is usually similar for all the productions. Some steps can be arranged differently, but the structure of the pipeline is always similar. Video game production, instead, is less linear, and one of the characteristics of the pipeline for this media is that it can be altered depending on the type of game that is being produced [Toftedahl and Engström, 2019]. Overall, the pipeline for video game production can be split into three different parts: the creation of the assets, the optimisation and conversion of the asset in an intermediate file format and, finally, the orchestration of the assets for a runtime environment [Lear et al., 2019].

For what concerns specific stages of the pipeline, as video games are an interactive media, the process is a bit different from the classic animation because the characters need to react to the input from the user. While the modelling, texturing and rigging stages of the 3D animation pipeline could be identical to those in the video game animation pipeline, the others need modification.

For what concerns the layout, the main difference is that the positioning of the characters in the scene might depend on the user input, so the artists do not always place the objects in the scene like in animation. The animation stage is very different in video games. Because of the interactivity of the medium, there cannot be a unique animation in place. Instead, in video games animation, the artists prepare *actions*, which is a set of animation assigned to a character or an object, that can be changed programmatically by the developers of the game to make the characters in the scene able to react to the user.

The VFX in video games are dynamic and usually scripted with certain events programmed by the developers. For what concerns the lighting, usually it is handled with a combination of techniques. Because the lighting is

heavy to compute, usually most of the lighting is baked, leaving dynamic lighting only to objects that the user can move. Recent technological advancements allow the use of full dynamic lighting with real-time ray-tracing. Finally, the rendering in video games is real-time, most commonly set to 30 or 60 frames per second.

2.5.3 Game engines in animation production

As mentioned in section 1.4, in the latest years several the adoption of a game engine for animation production is becoming more and more popular. Because of the novelty of the field, literature is still scarce. Epic Games, the developers of Unreal Engine, one of the most popular game engines in video game production, published a white paper on the importance of real-time rendering in film and television production [Epic Games, 2017]. As the title suggests, it focuses on the advantages of the integration of real-time technology in live-action products. Bąk and Wojciechowska [2019b], instead, describes a general approach to using game engines in animation production. They do not analyse a specific game engine but rather describe the traditional pipeline and the optimisations that a game engine can provide to it.

One work that describes an animation pipeline that involves a game engine is [Pohl et al., 2018], which employed Unreal Engine to produce the trailer for *Fortnite*. There are two main differences between their work and the one presented in this thesis. The first one regards the software used in the pipeline. As *Fortnite* is a game from Epic Games, the whitepaper in exam, of course, employs Unreal Engine, in addition to several other tools for the production of the assets. While the pipeline proposed in this thesis also employs Unreal Engine, it has been built to be modular, to be able to use different game engines or tools depending on the necessity.

The second difference is the communication between the different tools in the pipeline. The pipeline proposed by Epic Games uses UnrealGameSync, a tool firstly designed for game development but extended for the animation pipeline. This tool allows to sync and keep different versions of the assets; however, it is still strictly connected with an Unreal project and needs the Unreal Editor installed in the machine, even if the artists do not need to use it for their work. Moreover, the artists still need to manually export each asset and then load it in the UnrealGameSync tool to sync it. Instead, the pipeline proposed in this thesis introduces a dedicated communication layer between all the software involved in the pipeline. This communication layer is implemented as a python extension for each software and allows to load and save assets from any of this software with the same user interface. This

way, any software can access all the assets, loading the correct format in a completely transparent way for the users. The proposed communication layer also provides the versioning of the assets, allowing the user to easily update any loaded asset in the scene.

2.5.4 Summary

This section presented a brief explanation of the animation pipeline for what concerns 2D and 3D animation and the main differences in the production of animations for video games. It also analysed a few works about the employment of a game engine in animation production.

This section also highlighted the main differences of an existing pipeline that involves a game engine, proposed by Epic Games, with the pipeline proposed in this thesis, in chapter 4. The introduction of a game engine, in addition to all the advantages that provides to the animation production, allows the employment of the same assets for either animation and video games. Moreover, it also includes the methods, introduced in chapter 3, to generate 3D models from the 2D cartoons while maintaining their peculiar traits.

Chapter 3

Generation of 2.5D and 3D Characters from Existing 2D Cartoons

One of the main topics of this research project is the employment of 2D cartoon characters in a 3D context. This chapter introduces three methods that have been designed to generate a 3D approximation of of existing 2D cartoons, while focusing on maintaining the distinctive traits of the character in 3D as well.

Section 3.1 introduces a 2.5D modelling method, a first attempt to adapt the character to a 3D environment. However, the 2.5D approach has been discontinued in favour of a full 3D method, in section 3.2, with two variants. The first one, in section 3.2.1, is based on inflation and supported by a surface registration method. The second one, in section 3.2.2, which uses information from the side views to generate the model by solving an optimization problem. In the end, section 3.3 introduces a sketch-based posing system, which is employed to generate 3D animation from the 2D ones available for the 2D characters.

To conclude the chapter, section 3.4 analyses the results obtained and presents an evaluation of the methods with the support of the artists from Cloth Cat.

3.1 2.5D Modelling from Cartoon Characters

The first approach attempted to introduce 2D characters in a 3D environment employs 2.5D models. As 2.5D models consists of 2D shapes that move in a 3D

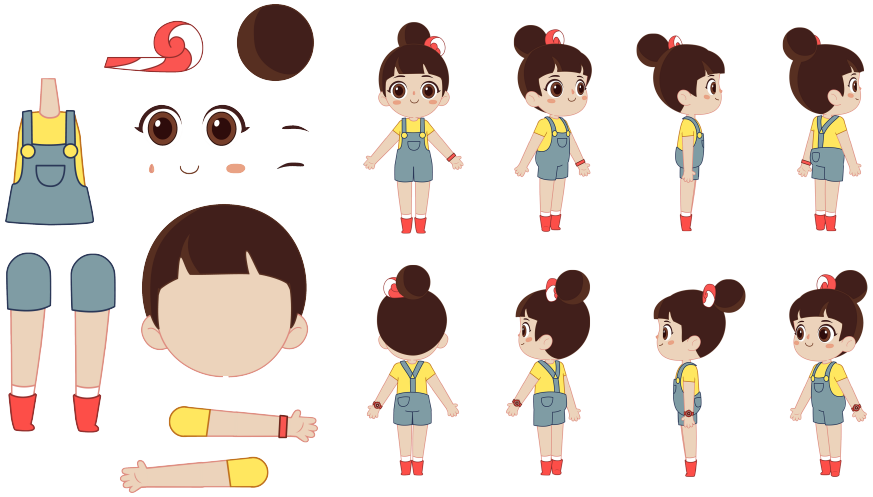


Figure 3.1: The input of the system. Every character is split up into different body parts, as on the left, and it is provided drawn from eight different perspectives, as on the right (Copyright ©, Magic Mall, Cloth Cat Animation).

space, they have been taken in consideration because they are the most immediate way to represents 2D objects in 3D. Differently from previous methods, described in section 2.1, which require to draw the character with their tool, this approach uses existing images of 2D cartoons, to be able to generate existing characters from previous 2D animation productions. The work presented in this section has been published in *Next generation computer animation techniques* [Barbieri et al., 2017].

The idea of this system is to simulate the depth between the different body parts of the character by moving them in the scene according to the position of the camera. To do this, it employs billboarding and parallax scrolling. Additionally, the system uses a 2D shape interpolation method to change the shape of the body parts while the camera rotates around the character, in order to simulate the real rotation by making appear new details that were not visible from the original perspective. As shown in figure 3.1, the input character is provided from the 2D animation pipeline in its turnaround and split into different body parts.

3.1.1 Billboarding

The body parts are placed all on the same plane $\mathcal{P}$, the billboard [Akenine-Moller and Haines, 2002], which will be rotated constantly facing the camera, as shown in figure 3.2. To determine the rotation matrix of the billboard which rotates around the y-axis, and with the viewer looking towards the negative

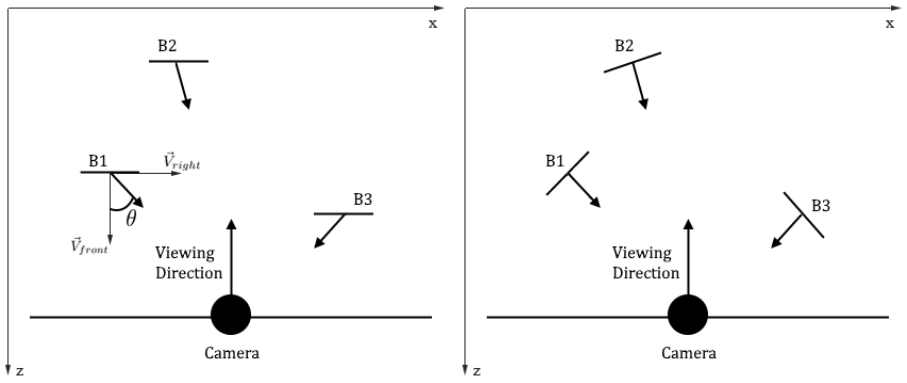


Figure 3.2: This example shows how the billboarding works. B1, B2 and B3 are the billboards, which always face the camera.

z-axis, the eye vector is computed from the model view matrix M:

$$\vec{V}_{eye} = M^{-1} \begin{pmatrix} 0 \\ 0 \\ -1 \\ 0 \end{pmatrix} \quad .$$

The rotation θ about the y-axis is then computed as:

$$cos\theta = \vec{V}_{front} \cdot (\text{eye}_{pos} - \text{obj}_{pos}) \ ,$$
$$sin\theta = \vec{V}_{right} \cdot (\text{eye}_{pos} - \text{obj}_{pos}) \ ,$$

where

$$\vec{V}_{front} = \Big(0, 0, 1\Big) \ ,$$
$$\vec{V}_{right} = \Big(1, 0, 0\Big) \ .$$

The rotation matrix R around the y-axis:

$$R = \begin{bmatrix} cos\theta & 0 & sin\theta \\ 0 & 1 & 0 \\ -sin\theta & 0 & cos\theta \end{bmatrix} \tag{3.1}$$

is then concatenated to the M matrix. The matrix MR is therefore used to transform the billboard geometry. A similar operation is used to rotate the billboard around the x axis too, however the rotation is attenuated by an arbitrary value ξ. Additionally, the rotation around this axis is limited. This limitation is necessary because in the animation pipeline the top view of char-

acter is not usually available for 2D productions.

3.1.2 Parallax Scrolling

On the billboard, the body parts move according to the movement of the camera and the parallax scrolling [Balkan et al., 2003]. While the camera turns around the character, the body components in the closest half of the character's body move in the camera's opposite direction, and the ones in the farthest half move in the same direction. The central part, such as the torso, however, does not move at all, except for the rotation due to the billboarding. Furthermore, according to parallax scrolling, the closest parts move faster. The reversed movement of the farthest component is explained by the *negative speed* of their movement, as they move *slower* than the closest parts. A layer λ is assigned to each body part of the character that should move when the camera is rotated. λ represents the proximity of the body part with the camera; a higher λ corresponds to a body part that is closer to the camera. The torso's λ is 0, so it will not move. For the closest parts, it will be positive, while negative for the farthest ones. When the camera is moved by a translation T_{camera}, the body parts will simply be moved by:

$$T_{bodypart} = \frac{T_{camera} \cdot \lambda}{\zeta} \ , \tag{3.2}$$

where ζ is an attenuating value to avoid that body parts move away from the others. This value depends the scale of the character. With the performed tests, the value that achieved the best results for a character of Unity scale 1 is 62.

λ is automatically computed by the system, by analysing the position of the body part in each view. Considering figure 3.3 as example, $\lambda = 0$ is defined as the middle point of the view by default, although it can be changed by the user. Based on the distance from this middle point, the value of λ grows or diminish, depending on the direction. In the example in the figure, the value of λ is being determined for the body parts in the front view, therefore λ will increase towards left — the direction of the front view — and will decrease towards right. For the rear view, the same values are used, but with the opposite sign. The body parts are assigned the maximum λ value — for the positive values, the minimum for the negative — that they reach, thus in the example the eyes and the mouth will be assigned a λ value of 1, while the chignon is assigned the value of -2.

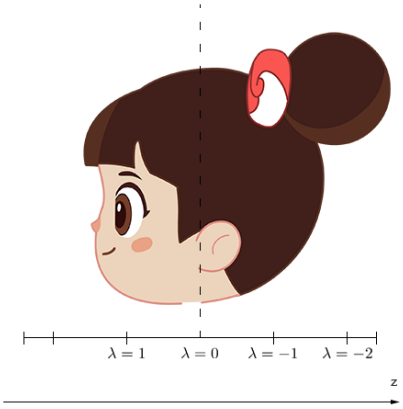


Figure 3.3: An example of how the λ value is assigned to each component. In this example the left side of the character's head is taken into account. The reported values are not assigned to the body parts on the left side, but on the front one, and on the rear one with opposite sign. (Copyright ©, Magic Mall, Cloth Cat Animation)

3.1.3 Shape Interpolation

Additionally to the billboard and parallax scrolling, while the camera rotates around the character, the system will automatically change the image of the body part from the turnaround, once the camera surpasses a certain threshold. To avoid an abrupt change of shape when the threshold is hit, the system employs a 2D shape interpolation method to produce a smooth transition. Figure 3.4 shows the different view angles and the thresholds.

The mesh $\mathcal{M}$ of the body part is divided into two sections. The *joint part* – *jp* – is the section that appears in both the meshes from the adjacent perspectives. The rest is the *margin*, defined as *appearing margin* – *am* – on the mesh of the perspective the camera is rotating to, and as *disappearing margin* – *dm* – on the mesh of the perspective the camera is rotating from. The difference between appearing and disappearing margin is based only on the rotation direction of the camera. Thus, they are not fixed as appearing or disappearing. The user is required to manually identify the joint part and the margin part of the mesh. Figure 3.5 shows an example of these sections.

The interpolation is applied only if the camera is between the two thresholds. The interpolation is defined as i^t, where $t \in [0, 1]$, $t = 0$ at the starting threshold and $t = 1$ at the ending threshold. Depending on the position of the camera, a certain i^t is computed and shown.

a is defined as the angle between the character viewing direction and the vector from the character's position to the camera's position, as illustrated by

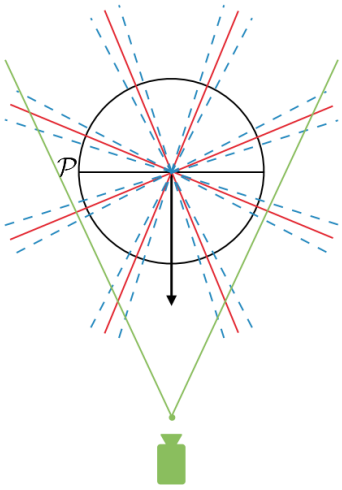


Figure 3.4: The character from the top view. The arrow is the viewing direction of the character. The red lines mark the points in which the perspective of the character changes. The dashed lines indicate the threshold for the interpolation.

figure 3.6, t is simply computed from a and the two thresholds:

$$t := \frac{a - t^0}{t^1 - t^0}, \tag{3.3}$$

where t^0 and t^0 are the angles of the first and second thresholds respectively.

Each interpolation i^t is a composition of two distinct operations. For what concerns the margins, indicating the width of the margin as m_{width}, for an interpolation i^t, the mesh $\mathcal{M}$ is cut along the y-axis at $x = m_{width} \cdot t$. If the margin is an appearing margin, the vertices on the closest part of the margin to the joint part will be added to the mesh. Instead, if it is a disappearing margin, the vertices on the farthest part of the margin to the joint part will be removed from the mesh. Whether these two parts are on the left or right part of the margin, it depends on the rotating direction of the camera. Figure 3.7 shows this procedure.

For what concerns the joint part of the shape, the system employs a shape interpolation method. A version of Bounded Distortion Harmonic (BDH) Shape Interpolation [Chien et al., 2016] has been implemented. This method produce "provably good"[Poranne and Lipman, 2014] harmonic mappings – they are smooth, locally injective and have bounded conformal isometric distortion – such that its geometric distortion is bounded by the input mapping's one. Moreover, this method does not employ mathematical optimization. As the system requires to compute the interpolation for each body part of each

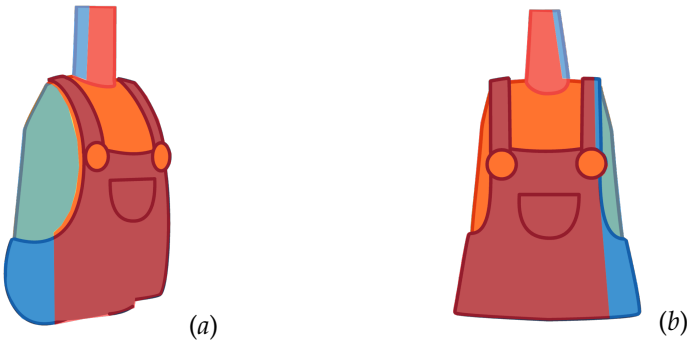


Figure 3.5: The same body part from two different perspectives. The red section of these two body parts is the joint part. The blue parts are the margins. (Copyright ©, Magic Mall, Cloth Cat Animation)

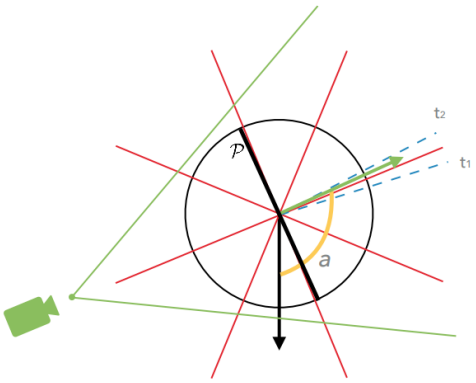


Figure 3.6: In this example, the camera is between the two thresholds, and the angle a is computed for the correct interpolation.

character every time the user moves significantly in the scene, the parallel capability of this method makes it the most appropriate for the framework.

Mathematical Background

To properly illustrate the 2D shape interpolation method, it is necessary to introduce some notions of complex analysis. Most of the notions included in this section are introduced in the book from Duren [2004], except where mentioned otherwise.

The first notion is the **complex derivative**. Considering a C^1 function $f : \Omega \to \mathbb{R}^2$ where $\Omega \subseteq \mathbb{R}^2$ is a domain. The Jacobian of the function, as well as any 2×2 real matrix, it can be written as the sum of a similarity and anti-

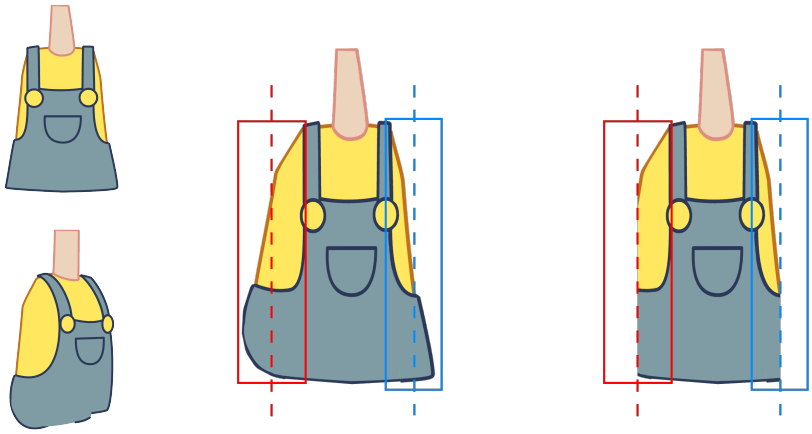


Figure 3.7: An example of the interpolation of the margins at $t = 0.5$. The first columns shows the original images. While the central part is the result of the deformation of the BDH interpolation, an intermediate state between the origin and target transformations, the other two sections, marked by the red and blue boxes, are the margins. As in this example $t = 0.5$, those sections are cut by half. (Copyright ©, Magic Mall, Cloth Cat Animation)

similarity matrix. Given two sets of matrices:

$$\begin{aligned} S &= \left\{ \begin{pmatrix} a & -b \\ b & a \end{pmatrix} \middle| \, (a,b) \in \mathbb{R}^2 \right\}, \\ A &= \left\{ \begin{pmatrix} c & d \\ d & -c \end{pmatrix} \middle| \, (c,d) \in \mathbb{R}^2 \right\}, \end{aligned} \tag{3.4}$$

the matrices in S apply a transformation by rotation and scaling and those in A apply the transformation after a reflection on the x axis. This decomposition of the matrix is defined as *additive decomposition*. As $\mathbb{R}^2$ can be represented as $\mathbb{C}$, it can be noticed that the matrices in S and A are the matrix notation of the complex numbers $z = a + \mathrm{i}b$ and its conjugate $\bar{z} = c + \mathrm{i}d$ respectively. According to [Ahlfors, 1953], given a complex function $f(x,y)$ of two real variables, by introducing the complex variable $z = x + \mathrm{i}y$ and its conjugate $\bar{z} = x - \mathrm{i}y$, $f(x,y)$ can be considered as a function of z and $\bar{z}$ if x and y are written as $x = \frac{1}{2}(z + \bar{z})$ and $y = -\frac{1}{2}(z - \bar{z})$. This leads to the definition of the Wirtinger derivatives:

$$\begin{aligned} \frac{\partial f}{\partial z} &= \frac{1}{2}\left(\frac{\partial f}{\partial x} - i\frac{\partial f}{\partial y} \right), \\ \frac{\partial f}{\partial \bar{z}} &= \frac{1}{2}\left(\frac{\partial f}{\partial x} + i\frac{\partial f}{\partial y} \right). \end{aligned} \tag{3.5}$$

For simplicity, these partial derivatives are indicated as $f_z := \frac{\partial f}{\partial z}$ and $f_{\bar{z}} := \frac{\partial f}{\partial \bar{z}}$. Using the additive decomposition for the Jacobian matrix, the complex numbers for the similarity and anti-similarity parts of the matrix: $f_z = a + \mathrm{i}b$ and its conjugate $f_{\bar{z}} = c + \mathrm{i}d$ are obtained.

The second notion introduced in this section is the **holomorphic and anti-holomorphic mapping**. Given a complex function $f(x + \mathrm{i}y) = u(x, y) + \mathrm{i}v(x, y)$, it is holomorphic if the function u and v have partial derivatives with respect to x and y and satisfy the Cauchy–Riemann equations:

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y} \quad \text{and} \quad \frac{\partial u}{\partial y} = -\frac{\partial v}{\partial x}. \tag{3.6}$$

A planar mapping can also be defined as holomorphic if $f_z = 0$ everywhere [Gunning and Rossi, 2009]. Conversely, an anti-holomorphic mapping is a planar mapping for which $f_{\bar{z}} = 0$ everywhere.

Next, it is introduced the notion of **harmonic planar mapping**. Given a function $u(x, y) : \Omega \to \mathbb{R}$, where $\Omega \subseteq \mathbb{R}^2$, it is harmonic if it satisfies the Laplace's equation:

$$\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0. \tag{3.7}$$

In an holomorphic function $f(x + \mathrm{i}y) = u(x, y) + \mathrm{i}v(x, y)$, its real and imaginary part are harmonic functions, where v is the harmonic conjugate of u, as each of these function is a solution to Laplace's equation, as they satisfy the Cauchy–Riemann equations [Evans, 1998]. Although it is not always true the opposite, on a simply connected domain Ω, an harmonic planar mapping f can be written as the sum of a holomorphic and an anti-holomorphic function:

$$f(z) = \Phi(z) + \overline{\Psi}(z), \tag{3.8}$$

where $\Phi(z)$ and $\Psi(z)$ are holomorphic functions. The parts of the additive decomposition of its Jacobian J_f can be integrated separately to obtain $\Phi(z)$ and $\Psi(z)$. For this mapping, $f_z = \Phi'$ is holomorphic and $f_{\bar{z}} = \overline{\Psi'}$ is anti-holomorphic.

The final concept introduced is the **local injectivity**. A continuous function $f : U \subset \mathbb{R}^n \to \mathbb{R}^n$ is locally injective at $x_0 \in U$ if exist a neighbourhood $V \subset U$ of x_0 such that $f|_V$ is injective. A mapping f is locally injective and sense-preserving — preserve the orientation — at a point z if $\det(J_f(z)) > 0$. The determinant of the Jacobian can be defined in terms of the Wirtinger's derivatives:

$$\det(J_f) = |f_z|^2 - |f_{\bar{z}}|^2. \tag{3.9}$$

Thus, a mapping f is locally injective and sense preserving if:

$$|f_z| > |f_{\bar{z}}|. \tag{3.10}$$

BDH Shape Interpolation

Now that the basic notions have been laid down, the method can be introduced. Given a domain $\Omega \subseteq \mathbb{R}^2$ and two locally injective sense-preserving harmonic mappings $f^0, f^1 : \Omega \to \mathbb{R}^2$, BDH shape interpolation produces an interpolating function $f : [0,1] \times \Omega \to \mathbb{R}^2$, where the interval $[0,1]$ corresponds to the rotation between the two perspectives. In the context of the system proposed in the thesis, the domain Ω is a 2D mesh, and the method aims to compute the interpolation $f|_{\{t\}\times\Omega}$ between the two perspectives f^0, f^1. The interpolation must satisfy the following properties:

- interpolation: $f|_{\{0\}\times\Omega} = f^0$ and $f|_{\{1\}\times\Omega} = f^1$;
- harmonicity: $f|_{\{t\}\times\Omega}$ harmonic $\forall t \in [0,1]$;
- locally injective: $f|_{\{t\}\times\Omega}$ is loc. inj. sense-preserving $\forall t \in [0,1]$;
- smoothness: $f|_{[0,1]\times\{z\}}$ is C^∞ $\forall z \in \Omega$.

For simplicity, for the rest of this section it is established that $f^t := f|_{\{t\}\times\Omega}$.

The idea of the method is to interpolate the Jacobian J_f and then to integrate it to obtain an interpolation. The decomposition in equation 3.8 shows that the similarity and anti-similarity part can be interpolated separately as $f_z = \Phi'$ and $f_{\bar{z}} = \overline{\Psi}'$. As they are holomorphic and anti-holomorphic, they are integrable in the the holomorphic and anti-holomorphic parts Φ and $\overline{\Psi}$, which summed give the final interpolated mapping.

For what concerns the similarity part f_z, it is obtained by linearly interpolating the logarithms of f_z^0 and f_z^1, referred as logarithmic interpolation. This quantity represents the linear interpolation of the angle of the closest rotation transformation to J_f at the point z and it is expressed as:

$$f_z^t = (f_z^0)^{1-t}(f_z^1)^t. \tag{3.11}$$

To determine the non-integer powers, it is necessary to introduce the complex logarithm. Given a complex number z, the complex logarithm is defined as the complex number $w = \log z$ for which $e^w = z$. As any complex number $z \neq 0$ has infinite complex logarithm, z is given in polar form as $z = re^{i\theta}$, and the logarithm is defined as $w_k = \ln(r) + i(\theta + 2k\pi) |\ k \in \mathbb{Z}$. The angle θ is the argument, denoted as $\mathrm{Arg}(z)$.

Defined the complex logarithm, the equation 3.11 can expressed as linear interpolation of logarithms:

$$\begin{aligned} f_z^t &= \exp\left((1-t)\log f_z^0\right) \cdot \exp\left(t \log f_z^1\right) \\ &= \exp\left((1-t)\log f_z^0 + t \log f_z^1\right) \\ &= |f_z^0|^{1-t} \cdot |f_z^1|^t \cdot \exp\left(\mathrm{i} \cdot (1-t)\operatorname{Arg}(f_z^0) + t \operatorname{Arg}(f_z^1)\right). \end{aligned} \tag{3.12}$$

For what concerns the anti-similarity part $f_{\bar{z}}$, the logarithmic interpolation cannot be used, as $f_{\bar{z}}^1$ and $f_{\bar{z}}^1$ vanish at points in Ω, and a logarithm for them cannot be defined. Instead, given a planar mapping f, two relevant quantities are introduced. The first is the Beltrami coefficient, also known as first complex dilatation, is defined as $\mu = \frac{f_{\bar{z}}}{f_z}$. The second is the second complex dilatation, is defined as $\nu = \frac{\overline{f_{\bar{z}}}}{f_z}$. The modulus of the Beltrami coefficient, $k = \frac{|f_{\bar{z}}|}{|f_z|}$ is called little dilatation, and it measures the conformal angle distortion. Having $k^t \leq \max(k^0, k^1)$ for all $t \in [0,1]$ is ideal to keep the conformal distortion bounded.

As the second complex dilatation satisfies $|\nu| = |\mu| = k$ and is holomorphic as long as f_z does not vanishes, it can be linearly interpolated as:

$$\nu^t = (1-t)\nu^0 + t\nu^1. \tag{3.13}$$

As ν^0 and ν^1 are holomorphic, ν^t is also holomorphic for all t, and the anti-similarity part of the equation 3.8 can therefore be obtained as:

$$f_{\bar{z}}^t = \overline{\nu^t f_z^t}. \tag{3.14}$$

Algorithm 1 Body part interpolation

```
1:  procedure InterpolateBodyPart(bodypart, angle)
2:      t ← (angle − t_0)/(t_1 − t_0)
3:      bodypart_jp ← BDH(bodypart, t)
4:      split margin at x_cut ← m_width · t
5:      if margin is am then                ▷ in case of an appearing margin
6:          increase size of the mask
7:      else                                ▷ in case of a disappearing margin
8:          reduce size of the mask
9:      end if
10: end procedure
```

Together, equations 3.11 and 3.14 provide the formulae to compute f_z^t and $f_{\bar{z}}^t$, which, integrated, produce the interpolated mappings, and, consequently, the *joint part* of the mesh at the t rotation of the camera. For what concerns

the implementation details, the system has been developed on the game engine Unity. The two operations, on the margins and on the joint parts, work in different ways. Considering the margins first, they are mesh generated by uniformly distributing points on the surface and then computing a constrained Delauney triangulation [Chew, 1989]. While the camera rotates, to cut the margin without the need to edit the mesh, a mask has been employed, to hide the section of the mesh that it is not required only. The meshes of the margins are placed in a plane slightly closer to the camera, in order to appear over the joint part. For what concerns the joint part, instead, the method of R. Chen and Weber [2015] has been employed to generate the harmonic mappings which are the input for BDH shape interpolation method employed. Algorithm 1 summarizes the whole procedure.

3.1.4 Combining the three techniques

The parallax scrolling works on a higher level. In fact, while the interpolation operates distinctly for each body part, and only if the camera is between two thresholds, the parallax scrolling constantly moves all the body parts of the character. To each body part, as the camera moves in the scene, is applied the translation in equation 3.2.

As the character is a 2D figure, it lies on a 2D plane. The transformations applied by both the parallax scrolling and the shape interpolation do not add a third dimension to the mesh's vertices. Thus, the billboarding is just applied to the character's plane, and not individually to each component.

Algorithm 2 Camera's movement handler

1: **procedure** OnCameraMovement
2: $\vec{u}$:= vector between character's position and camera's position
3: $\vec{v}$:= character's viewing direction
4: $angle \leftarrow \arccos(\vec{u} \cdot \vec{v})$
5: **for all** *bodypart* in *character* **do**
6: **if** $t_1 \leq angle \leq t_2$ **then**
7: InterpolateBodyPart(bodypart, angle)
8: **end if**
9: $bodypart_{position} += (camera_{translation} \cdot \lambda)/\zeta$
10: **end for**
11: rotate the character's plane around its central position towards the camera
12: **end procedure**

The whole procedure is performed every time the camera moves, although the interpolation is executed only if the camera is in specific positions, in particular between two thresholds. Algorithm 2 outlines the entire operation.

(*a*) (*b*) (*c*) (*d*)

Figure 3.8: Two examples of 2D characters placed in a 3D environment using the framework. (*a*), (*b*) Copyright ©, Esoteric Software. (*c*), (*d*) Copyright ©, Toot Enterprises Limited.

Figure 3.8 (a) and (b), as well as figure 3.10, show how the different body parts move at the same time of the camera, because of the parallax scrolling. The boy in figure 3.8 has the right arm and leg, as well as an eye and the mouth, in a front layer, hence the λ in the equation 3.2 is positive. The left arm and leg, instead, has a negative λ, as they are in the hindermost part of the character. In the end, the λ value for the torso and the head is 0. In the example in figure 3.8 the camera is moving around the character in a counter clockwise sense – i.e. from the left to the right – therefore the body parts with positive λ are moving from the right to the left, in the opposite direction, while the ones in the rear side move in the same direction as the camera. Moreover, as the forearm is closer to the camera compared to the upper arm, it has a greater λ, therefore, it is moved more on the left.

Figure 3.9 shows instead an example of the interpolation of a body part between two different perspectives. Figure 3.9(*a*), the source, and figure 3.9(*b*), the target, show the body parts from which the interpolation is computed. These two images represent the same body part, but from two different perspectives. In the two figures are also highlighted the joint part, in red, and the margins, in blue. In figure 3.9(*c*), the joint part is interpolated using BDH shape interpolation, and the two margins are added to the body part. In the figure, the appearing margin is surrounded by the red box, while the disappearing margin by blue box. The dashed lines mark the section of the margins that will be removed. In this particular example, $t = 0.65$. Figure 3.9(*d*) represents the final result of the computation. It can be noticed that there is some residue in the appearing margin, to the left of the dashed line. That is because that section of the hair is part of the joint section, thus it is not removed.

In the end, figure 3.10 show a complete transaction of the camera from the front-right perspective to the front one. It can be noticed in figure 3.10(*b*), in particular on the torso and the head, the interpolation of the two states. It can

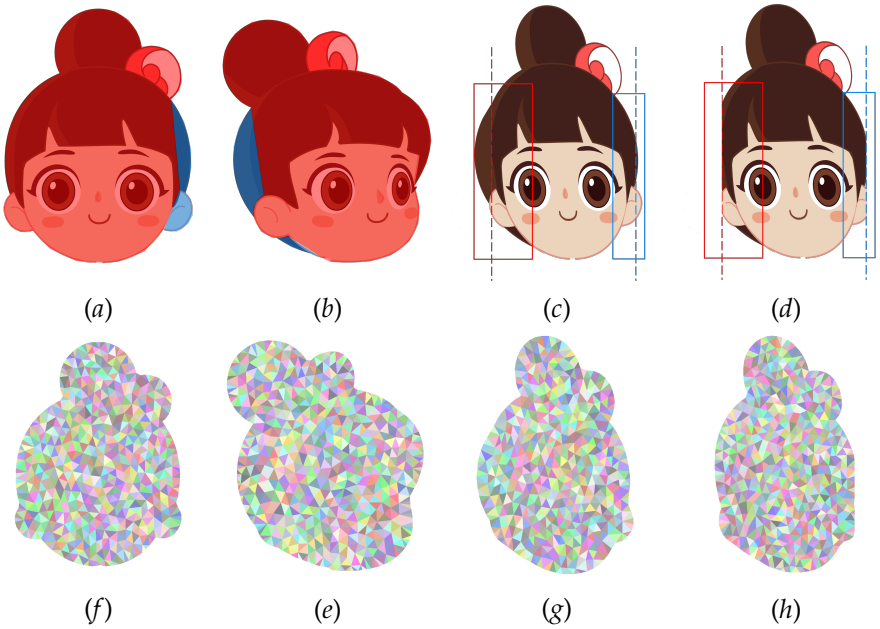


Figure 3.9: An example of the interpolation process. In (*a*) and (*b*), the red sections are the joint parts, while the blue ones are the margins. (*c*) shows the interpolated joint part merged with the two margins, while in (*d*) the margins are cut and it shows the final interpolation at $t = 0.65$. It must be noticed that the deformation is on the mesh, not on the texture, which is just applied on it. In the second row, in (*e*), (*f*), (*g*) and (*h*), the deformation process on the meshes is shown. (Copyright ©, Magic Mall, Cloth Cat Animation)

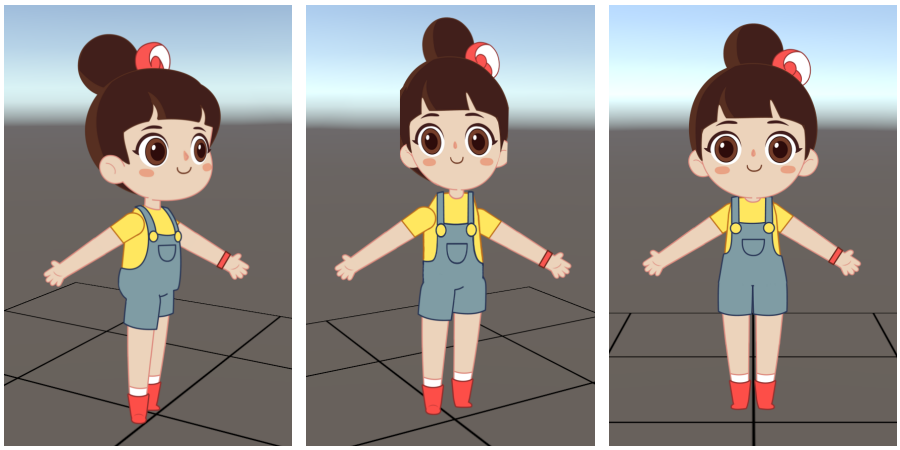
be also noticed the parallax scrolling on the arms and the legs.

The tool is implemented as a Unity3D native plug-in. This allowed the employment of Nvidia's CUDA to parallelize on the GPU as many operations as possible, in particular the interpolation.

3.2 3D Modelling from Cartoon Characters

While the system presented in the last section has some perks compared to similar 2.5D methods, in particular its ability to work with pre-existing cartoons instead of the requirement of creating ad hoc characters, it has been decided to change approach and adopt a full 3D solution.

There are several advantages with a 3D solution, compared to a 2.5D one. First of all, this kind of approach admits more flexibility with the camera. With the previous method, the camera is limited vertically because of the flatness of the character, which would require an extra input image from the top view for each body part to interpolate the shape. The objective of the thesis is to repurpose existing cartoons to be used in a 3D environment, and while a



(*a*) (*b*) (*c*)

Figure 3.10: A complete example of the system. (*a*) shows the character from the front-right perspective, while in (*b*), it is in the middle of the transaction between that state and the front. Lastly, (*c*) shows the character from the front. (Copyright ©, Magic Mall, Cloth Cat Animation)

turnaround of the character is commonly available in animation production, the same is not true for the top view. Even without an image for the top view, a 3D model generated from the turnaround still allows a free movement of the camera, as the character will not appear flat as in the 2.5D version. The second reason for turning to a 3D solution is to avoid some of the limitations of the 2.5D method, in particular to avoid the deformations that may appear during the interpolation between two body parts. The limitations of the previous method are discussed in more details in section 3.4. Finally, a 3D solution makes it straightforward the interaction of the character with its surrounding 3D environment and 3D objects in the scene.

This section will describe two methods for the generation of 3D models from 2D cartoons. The first one is based on inflation, in section 3.2.1, and it is complemented by a surface registration method, that allows the interpolation between models of adjacent points of view. The second one is a solution which uses information from the side views to create a more accurate model, by solving an optimization problem, in section 3.2.2.

Although the method that is presented in this chapter is 3D, it aims to meet all the points addressed in the introduction. As the generation of the 3D components of the character is based on its turnaround, the generation of the assets is still faster than modelling the 3D character from scratch. For what concerns the performance, the generation of the mesh is obtained in average a few seconds. The animation will be discussed in section 3.3.

3.2.1 Modelling through inflation

This section introduces the first method for the generation of 3D characters from 2D cartoons. This method is based on inflation and complemented by a surface registration method. The work described in this section has been published as a poster at SIGGRAPH 2018 [Barbieri et al., 2018].

The idea is to first generate a flat 2D mesh from each view of the turn-around, which then is inflated based on the distance from the contour. Then, a registration method is used to align two models generated from adjacent viewing directions and, while the camera rotates around the character, the position of the vertices is interpolated between the two registered states.

The purpose of the registration part is to keep the distinctive traits of the 2D character in 3D. In fact, these details are either entirely hidden in one specific view or one of their dimension is not explicit. While interpolating between two views, this method simulates the three-dimensionality of these features, as they appear while the camera rotates.

Inflation

To produce one of these models, first a 2D mesh is generated for each body part of the character. A number U of uniformly distributed points is placed inside the bounding box of the shape:

$$U = (r \cdot c) \cdot \rho, \tag{3.15}$$

where r is the number of pixel in a row, c is the number of pixel in a column and ρ is the density of the uniformly distributed points. Then, a minimum distance ε is set between the points:

$$\varepsilon = \frac{r + c}{2 \cdot \sqrt{U - 1}}. \tag{3.16}$$

The system computes the contour of the body part, which is then sampled to obtain only a limited number of points. The sampling is done by applying the equation 3.16 for the minimum distance between the points. A constrained Delaunay triangulation [Chew, 1989] is then computed to obtain the flat mesh. The system then inflates the flat mesh. Considering the section of the inflated model as an ellipse with width $2a$ and height $2b$, its equation is:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1.$$

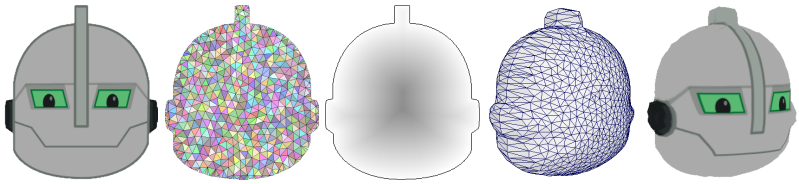


Figure 3.11: In order, are shown the original drawing of the head of the robot, the 2D mesh generated from the drawing, the computed distance map and the inflated mesh. (Copyright ©, Thud Media)

a can be considered the maximum distance from the contour $D_{\max}$, and b as the maximum desired inflation $H_{\max}$. Thus, for the x vertex of the mesh, its inflation is defined as:

$$H(x) = H_{\max} \cdot \sqrt{1 - \frac{(D_{\max} - D(x))^2}{D_{\max}^2}}, \tag{3.17}$$

where $D(x)$ is the distance of the vertex x from the contour. These distances are precomputed as distance transform [Felzenszwalb and Huttenlocher, 2012]. The inflated mesh is then mirrored to the back to obtain a closed 3D mesh. The steps of this procedure are shown in figure 3.11.

The generated 3D components are then combined to form the complete 3D character. If the character has been provided drawn from multiple points of view, along with the 2D skeleton for each point of view, it is possible to place correctly each body part in 3D. If only one image – and one 2D skeleton – is available, the system is not able to automatically place the components on the z axis, thus the user input is required.

Surface registration

At this point, a 3D model has been generated for each body part and for each provided perspective. In order to maintain all the character's features visible from different perspectives, a novel non-isometric surface registration method is proposed in this section, inspired by the works of [Jiang et al., 2017; I.-C. Yeh et al., 2011; Sorkine and Cohen-Or, 2004]. The registration of the surface between two meshes generated for adjacent perspectives will be employed to fit one into the other, and interpolate the vertices from one to the other — depending on the rotation of the camera — while the camera rotates around the character. This will help maintain features of the 2D character in 3D as well. The difference of this method from the ones cited earlier is in the input. In fact, as the input consists of meshes generated from cartoon characters, which is manually drawn by the artists from their 3D imagination

and perception, it will be not accurate as real projections, so it is impossible to use previous computer vision research directly. The result will be matching the two meshes as close as possible.

Given a template — or source — mesh $\mathcal{S}$ and a target mesh $\mathcal{T}$, which are mesh generated from two adjacent perspectives, and a set of feature points to mark correspondence between the two meshes, the goal is to deform the template $\mathcal{S}$ gradually into $\mathcal{S}'$ so that the deformed mesh $\mathcal{S}'$ is sufficiently close to the target $\mathcal{T}$ with structure preserved and feature points matched. Let p, p', q denote the vertex positions on $\mathcal{S}, \mathcal{S}', \mathcal{T}$ respectively, the registration energy is defined as:

$$E(p') = w_l E_l(p') + w_d E_d(p') + w_c E_c(p') + w_f E_f(p'), \tag{3.18}$$

where E_l is the bi-Laplacian energy, E_d is the consistent as-similar-as-possible (CASAP) energy, E_c is the correspondence constraint energy, and E_f is the feature point constraint energy. The weights before each energy term adjust the influence they account for in the total energy. These energy terms are introduced from both mathematical and practical perspective.

The bi-Laplacian energy is defined as follows:

$$E_l(p') = \|Lp'\|_F, \tag{3.19}$$

where the matrix L is the discrete Laplace-Beltrmi operator, and $\|\cdot\|_F$ denotes the Frobenius norm. This term is introduced because the original mesh is almost evenly distributed. Minimizing this energy leads to a smooth mesh which evenly distribute the vertices fairness on the surface, as each vertex strives to locate in the centroid of its one-ring neighbours and thus the edge lengths strive to be locally equalized.

The CASAP energy is expressed by:

$$\begin{aligned} E_d(p') = \sum_i \Big(& \sum_{(j,k)\in\mathcal{E}_i} w_{jk} \cdot \zeta(i,j,k) + \\ & + \alpha A \sum_{\mathcal{E}_l \in \mathcal{N}(\mathcal{E}_i)} w_{il} \cdot \eta(i,l) \Big), \end{aligned} \tag{3.20}$$

where:

$$\begin{aligned} \zeta(i,j,k) &= \|(p'_j - p'_k) - s_i \mathbf{R}_i (p_j - p_k)\|^2; \\ \eta(i,l) &= \|\mathbf{R}_i - \mathbf{R}_l\|_F^2; \end{aligned}$$

$\mathcal{E}_i$ is the edge set which consists of the edges incident to vertex i and the

edges in the link of vertex i in the template mesh $\mathcal{S}$; scale factor s_i and rotation $\mathbf{R}_i$ describe the deformation belongs to vertex p_i; w_{jk} are edge weighting coefficients, chosen to achieve a consistent discretization of a membrane like energy [Pinkall and Polthier, 1993]; $\mathcal{N}(\mathcal{E}_i)$ are the neighbouring edge set of $\mathcal{E}_i$; α is a weighting coefficient; A is the area of the whole mesh surface, which is used to make the energy scale invariant; w_{il} are scalar weights. This energy constrains the deformation consistently in an as-similar-as-possible way.

Before formulating the correspondence constraint energy, it is necessary to find out the correspondences between the template and the target. For each vertex p_i' on the deformed template $\mathcal{S}'$, the closest point q_j on the target $\mathcal{T}$ is chosen as its correspondence at first, then the pairs are pruned out if the distance between them exceeds a threshold D or if the angle between their normal vectors exceeds a threshold Θ. Assuming the vertex $q_{idx(i)}$ is the correspondence of vertex p_i, $Proj(q_{idx(i)})$ is the projection of $q_{idx(i)}$ onto q_j's normal vector, then the correspondence constraint energy can be represented as:

$$E_c(p') = \|\mathbf{C}_c p' - Proj(\mathbf{D}_c q)\|_F^2, \tag{3.21}$$

where p', q are the vertex positions on surface $\mathcal{S}', \mathcal{T}$ respectively, and $\mathbf{C}_c, \mathbf{D}_c$ are the sparse matrices that define the pruned matching correspondences between $\mathcal{S}'$ and $\mathcal{T}$. Assuming the m-th correspondence is p_i on $\mathcal{S}'$ and $q_{\mathrm{idx}(i)}$ on $\mathcal{T}$, then

$$\mathbf{C}_c(m,n) = \begin{cases} 1, & \text{if } n = i \\ 0, & \text{if } n \neq i \end{cases},$$

$$\mathbf{D}_c(m,n) = \begin{cases} 1, & \text{if } n = \mathrm{idx}(i) \\ 0, & \text{if } n \neq \mathrm{idx}(i) \end{cases}.$$

The feature point constraint energy aims to attract the feature points on the template to the correspondent feature points on the target. This energy can be similarly defined as the correspondence constraint energy:

$$E_f(p') = \|\mathbf{C}_f p' - \mathbf{D}_f q\|_F^2, \tag{3.22}$$

where $\mathbf{C}_f, \mathbf{D}_f$ are the sparse matrices that define the feature point pairs between $\mathcal{S}'$ and $\mathcal{T}$.

To solve the deformed vertex position p', the method takes the derivative of the total energy 3.18 with respect to p', which produces the following linear system:

$$\mathbf{A}^T \mathbf{A} p' = \mathbf{A}^T b, \tag{3.23}$$

where

$$\mathbf{A} = \begin{pmatrix} w_l \mathbf{L}^T \mathbf{L} \\ w_d \mathbf{L} \\ w_c \mathbf{C}_c \\ w_f \mathbf{C}_f \end{pmatrix}, \; b = \begin{pmatrix} 0 \\ w_d d \\ w_c Proj(\mathbf{D}_c q) \\ w_f \mathbf{D}_f q \end{pmatrix},$$

$$d = \frac{1}{3} \sum_{j \in \mathcal{N}(i)} \Big(w_{ij}(s_i \mathbf{R}_i + s_j \mathbf{R}_j + s_m \mathbf{R}_m) + \\ + w_{ji}(s_i \mathbf{R}_i + s_j \mathbf{R}_j + s_n \mathbf{R}_n) \Big)(p_i - p_j).$$

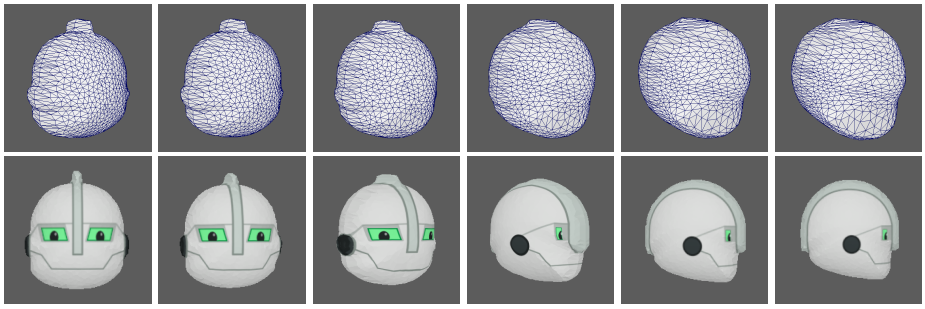


Figure 3.12: The transformation of a robot when the camera moves from the front to the right side of the character. The first row shows the transformation of the mesh from a fixed angle. In the second row, the texture has been applied to the mesh, and the transformation is seen from the camera point of view. The texture changes at a certain threshold, which in this case is in the middle of the transformation. (Copyright ©, Thud Media)

The registration is computed between all the pair of models generated from adjacent perspectives in both ways: using one as the template and the other as the target, and vice versa. In the system, the model of the closest perspective is shown. While the camera rotates around the character, the position of the vertices is linearly interpolated between their position in the source model and in the target. Figure 3.12 shows the transformation of the mesh between two adjacent perspectives from the camera view.

The correspondences between the two adjacent perspectives are computed before converting the images in 2D meshes. In the current 2D animation pipeline, the body parts are separated, and feature points such as the eyes or the mouth can be easily detected. Additionally, the animators can provide additional feature points. During the generation of the mesh, the system automatically places a vertex in those point. Figure 3.13 shows the two overlapping models with their feature points.

To each body part of the model is applied as texture the image from which

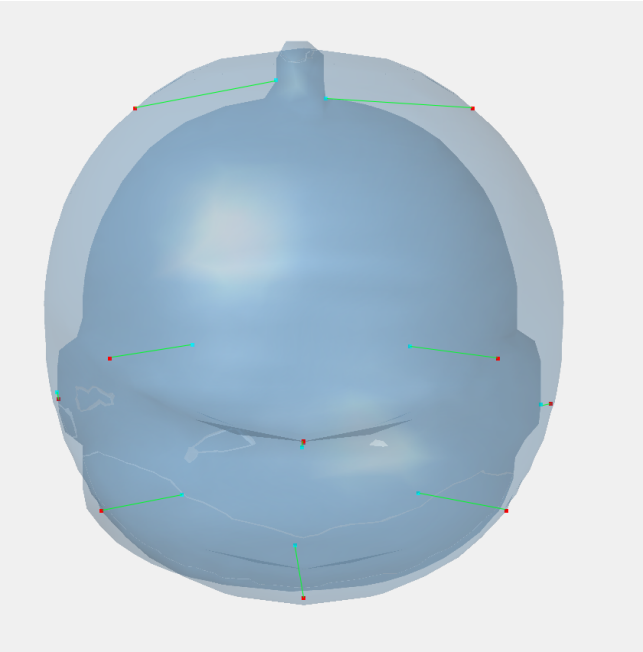


Figure 3.13: The correspondences between the two meshes: the mesh generated from the front image, in cyan, and the one generated from the right image, in red. The green line shows the correspondence between the points. (Copyright ©, Thud Media)

it has been generated originally. To reflect the change of the model as the camera moves around the character, a certain threshold is set, after which the texture of the model changes to the image of the body part from the new perspective.

3.2.2 Modelling through optimization

The previous method is able to achieve its purpose of generating a 3D character from a 2D cartoon, and to keep its distinctive traits using the surface registration method and interpolating between two adjacent views. However, an alternative method has been investigated that could embed the traits of the character directly in the 3D model, without requiring registration and interpolation. The focus of this variant method has been on cartoon heads. In cartoons, it is usually the face the body part with most of the unique traits that give personality to the character. The great variety of characteristics that could be present in all the body parts has led to the decision to focus initially on just the heads.

While drawings are an accessible and intuitive way to portray characters, they do not offer an accurate representation of reality. For this reason accurate methods for 3D reconstruction from, for instance, photographs or videos, can-

not be used for cartoons. However, in animation production, the turnaround of the characters is commonly available, and the drawings of the side views still convey information about features that might be occluded or partially visible in the front view, although not as accurate as in a photo. The method presented in this section, therefore, uses the turnaround of the cartoon character to generate an approximation of its 3D version as close as possible to the original drawing by solving an optimization problem.

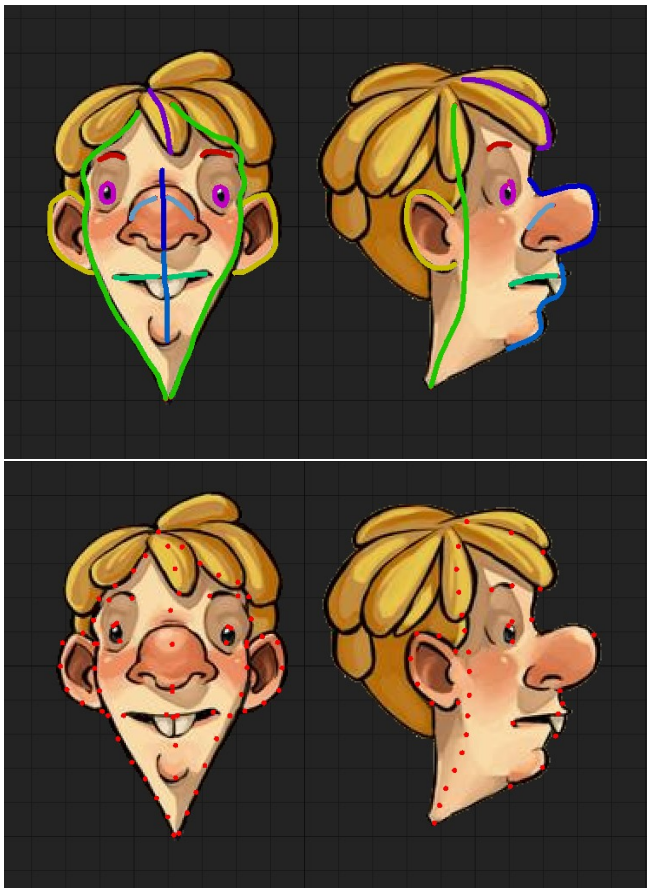


Figure 3.14: The feature points marked with the sketches. The second images shows the sampled feature points. The sampling rate can be set by the user. (Copyright ©, Caitlyn Patten)

As 2D cartoons do not always have evident landmarks, or they can be strongly altered in alternative views, the system avoids to try to identify automatically potential correspondences between the two input view, but it asks the user to input them. To improve the user experience, a sketching interface has been designed to allow the users to input the correspondence points in an easier and faster way, and the system will sample those to get two sets of

points — one for each view. Figure 3.14 shows the feature points sketched and sampled.

Given a front and a side of the character and two sets of n correspondence points $\mathbf{x}$ and $\mathbf{y}$, the goal is to generate a 3D model for the character by using information from both the images. The system computes a 2D mesh $\mathcal{F}$ and $\mathcal{S}$ respectively for front and side view of the character using the same method described in the previous section, however adding the points in $\mathbf{x}$ and $\mathbf{y}$ as constraint points for the triangulation. Instead of inflating the surface $\mathcal{F}$, in this case the system exploits the information embedded in the surface $\mathcal{S}$ to deform $\mathcal{F}$.

The energy function is defined as follows:

$$E = E_h + E_c + E_s \tag{3.24}$$

where E_h is the Laplacian energy, E_c penalize the distance between the feature points in the front and side views, and E_s favours deformations that follow the silhouette of the side view.

Laplacian Energy

The first term of the energy function 3.24 is a Laplacian energy. The objective is the computation of the influence $h_j : \mathcal{F} \to \mathbb{R}$ of the feature points $\mathbf{x}$ over the vertices of the mesh $\mathcal{F}$, and employing the Laplacian energy to find these influences guarantees that the solution are high-order, shape-aware and smooth functionals [Jacobson et al., 2012]. Before discussing about the specific case of this thesis, the problem should be introduced in a more general way.

Given a domain $\Omega \subset \mathbb{R}^n$ and a function $u : \Omega \to \mathbb{R}$, the Dirichlet energy is defined as:

$$\frac{1}{2} \int_\Omega \|\nabla u(x)\|^2 dx \tag{3.25}$$

This energy measures how variable the function u is. Solving the Laplacian equation $-\Delta u(x) = 0 \ \forall \ x \in \Omega$ is equivalent to minimize this energy. Let $\Omega \subset \mathbb{R}^2$ be the domain. The influence of the feature points on the domain determines its deformation. Given the transformation T_j for each feature point $\mathbf{x}_j$, the points $\mathbf{p} \in \Omega$ are deformed as:

$$\mathbf{p}' = \sum_{j=1}^{n} h_j(p) \cdot T_j \cdot \mathbf{p} \tag{3.26}$$

where n is the number of feature points.

The Laplacian energy E_h is defined as follow:

$$\min_{h_j} \sum_{j=1}^{n} \frac{1}{2} \int_{\Omega} \|\Delta h_j\|^2 dV \tag{3.27}$$

Inspired by the work of Jacobson et al. [2011], the following constraints are applied to this equation:

$$h_j|_{\mathbf{x}_k} = \delta_{jk} \tag{3.28}$$

$$\sum_{j=1}^{n} h_j(\mathbf{p}) = 1 \qquad \forall\ \mathbf{p} \in \mathcal{F} \tag{3.29}$$

$$0 \leq h_j(\mathbf{p}) \leq 1,\ j = 1, \ldots, n, \qquad \forall\ \mathbf{p} \in \mathcal{F} \tag{3.30}$$

The constraint 3.28 enforces each feature point to have maximum influence over itself but zero over all the others. δ_{jk} is the Kronecker's delta, which is defined as:

$$\delta_{ij} = \begin{cases} 0 & \text{if } i \neq j, \\ 1 & \text{if } i = j. \end{cases}$$

The constraint 3.29 enforces the sum of all the influences for every point in the mesh to be one. The constraint 3.30 enforces the value of the influences of the feature points to be between one and zero, also implying their non-negativity.

The equation 3.27 possess several desirable properties which are fundamental to produce the required results. The first property is the **smoothness** of the influence functions. A non smooth function would cause artifacts when deforming the mesh. The functionals are C^1 at the feature points, and C^∞ everywhere else, because of the fundamental lemma of calculus of variations. In fact, the lemma says that if a continuous function f on an open set $\Omega \subset \mathbb{R}^d$ satisfies the equality:

$$\int_{\Omega} f(x)h(x)dx = 0$$

for all compactly supported smooth functions h on Ω, then f is identically zero [Jost et al., 1998].

The second property is the **non-negativity**, enforced by the constraint 3.30. Negative influences over the points of the mesh are avoided to prevent deformation contrary to the direction of the transformation of the feature points. The same constraint also allows the functionals to **avoid local maxima**. The constraint forces the functional to have the global maximum on the feature points, as value of one, and do not have any other local maxima. The employment of the Laplacian operator allows the functionals to be **shape-aware**, as the influences h depends only on the metric of the mesh $\mathcal{F}$, differently from

the application of the Euclidean distance. The final property is the **partition of unity**, enforced by the constraint 3.29. In fact, that constraint implies that if all the feature points are transformed by the same amounts, the entire mesh will undergo the same transformation. Because of the constraint 3.30, this was not guaranteed, therefore it is enforced with an additional constraint.

Now that the problem has been stated, it will be discussed the strategy for its discretization. The domain Ω is discretized as the triangle mesh $\mathcal{F}$ introduced earlier, with m vertices, while the influences are denoted as piece-wise linear functions, laid out as column vectors $\mathbf{h}_j = (h_{(1,j)}, h_{(2,j)}, \cdots, h_{(m,j)})^T$ representing the influence of one feature point over all the vertices.

For what concerns the energy 3.27, the Finite Element Method (FEM) is employed for the discretization. According to the FEM, a discrete Laplacian can be represented as a $n \times n$ matrix $\mathbf{L}$ defined as:

$$L_{ij} = \begin{cases} \cot \alpha_{ij} + \cot \beta_{ij} & \text{if } \ j \in N(i), \\ 0 & \text{if } \ j \notin N(i), \\ -\sum_{k \neq i} L_{ik} & \text{if } \ i = j, \end{cases}$$

where $N(i)$ are the vertices adjacent to the vertex i and α_{ij} and β_{ij} are the angles opposite to the edge ij. Additionally, the FEM uses another matrix, the mass matrix $\mathbf{M}$, as the discretization of the inner-product. In other words, the mass matrix considers the area around each vertex. It is a $n \times n$ diagonal matrix, where each value M_{ii} is the Voronoi area around vertex i of the mesh. The FEM uses its inverse to transform integrated quantities into point-wise quantities. Hence:

$$\Delta h \approx \mathbf{M}^{-1}\mathbf{L}\mathbf{f}.$$

Therefore, the Laplacian energy 3.27 is discretized as:

$$\begin{aligned} \sum_{j=1}^{n} \frac{1}{2} \int_{\Omega} \|\Delta h_j\|^2 dV \ &\approx \ \sum_{j=1}^{n} \frac{1}{2} (M^{-1}L\mathbf{h}_j)^T M (M^{-1}L\mathbf{h}_j) \\ &= \ \frac{1}{2} \sum_{j=1}^{n} \mathbf{h}_j^T (LM^{-1}L)\mathbf{h}_j. \end{aligned} \tag{3.31}$$

Constraints Energy

The second term of the energy function 3.24 is a constraint for the feature points, and provides information to the energy about the side view of the character. The side view offer information about the depth of the points, information that cannot be retrieved with the front view image alone. As the side view mesh $\mathcal{S}$ is a 2D mesh, only the x coordinate is necessary, and the

method needs to reduce its the distance from the z coordinate of the correspondent feature point in the front mesh $\mathcal{F}$. Denoting with $\mathbf{x}_z$ the column vector of the z coordinates of the feature points in the $\mathcal{F}$ mesh, and with $\mathbf{y}_x$ the column vector of the x coordinates of the feature points in the $\mathcal{S}$ mesh. The constraint energy E_c is defined as:

$$\min_{\mathbf{x}} \ \|\mathbf{x}_z - \mathbf{y}_x\|^2 \tag{3.32}$$

The vertices on the mesh $\mathcal{F}$ are then computed according to equation 3.26.

Silhouette Energy

The final term of the energy function 3.24 is focused on retrieving more details from the side view. While the energy 3.32 computes an optimal z coordinate to the feature points, there are components that require more complex transformations of these points to create an appropriate model. In particular, occluded areas in $\mathcal{F}$ cannot be accurately represented with just energy 3.32.

Inspired by the work of Hahn et al. [2015], in this section the idea of the *sketch abstractions* — defined by the authors as a composition of rigged curves that form a 2D representation of the character from a particular point of view — has been extended to the modelling, introducing the following contributions. First of all, the method still uses 2D sketch, but they are actually curves in the 3D space. In [Hahn et al., 2015], the sketch abstraction is projected to obtain 2D points, and the target sketch is 2D as well. As the system works with images from different perspectives, it is convenient to keep the sketches as 3D curves. The second contribution is that the sketch abstraction and the target sketch are sampled with the same number of points. This way, an entire minimizations stage to compute the correspondence between the two samplings can be avoided. The final contribution regards the input of the energy function. As the purpose of this system is the modelling of a character, it does not have any rig. Instead, the method employs the feature points $\mathbf{x}$ to deform the mesh $\mathcal{F}$, with the influences computed with the energy function 3.27. The deformation of the points $\mathbf{p}$ of the mesh described by the equation 3.26 can be easily assimilated to a classic linear blend skinning.

Given a sketch $\mathbf{z} = (\mathbf{z}_1, \mathbf{z}_2, \cdots, \mathbf{z}_l)$ sampled in l points, in order to compute the i-th point of the sketch abstraction, the system projects the point on the surface of the mesh and pick the three closest vertices $\mathbf{p}_a^i$, $\mathbf{p}_b^i$ and $\mathbf{p}_c^i$. Then, it computes the barycentric coordinates α_i, β_i and γ_i for the point $\mathbf{z}_i$ with respect to the triangle composed by the three points picked previously. The point $\mathbf{z}_i$

is expressed in function of the feature points $\mathbf{x}$ as:

$$\mathbf{z}_i(\mathbf{x}) = \alpha_i \cdot \mathbf{p}_a^i(\mathbf{x}) + \beta_i \cdot \mathbf{p}_b^i(\mathbf{x}) + \gamma_i \cdot \mathbf{p}_c^i(\mathbf{x}) \tag{3.33}$$

The objective of this energy is to find the configuration of the feature points $\mathbf{x}$ such that it minimizes the function. Given a second sketch $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_l)$, sampled with the same number of points as $\mathbf{z}$, the silhouette energy E_s is defined as:

$$\min_{\mathbf{x}} \sum_{i=1}^{m} \|\mathbf{y}_i - \mathbf{z}_i(\mathbf{x})\|^2 + R_p + R_d \tag{3.34}$$

To avoid the the function to be underdetermined, two regularization terms are introduced. The first term promotes solutions with the least amount of change in the feature points positions:

$$R_p(\mathbf{x}) = \|\mathbf{x} - \mathbf{x}_0\|^2 \tag{3.35}$$

The second term penalize solutions that deform the mesh $\mathcal{F}$ in regions distant from the sketch $\mathbf{z}$.

$$R_d(\mathbf{x}) = \sum_{i=1}^{n} \frac{d_i}{d_{max}} \|\mathbf{p}_i(\mathbf{x}) - \mathbf{p}_i^0\|^2 \tag{3.36}$$

where $\mathbf{p}_i(\mathbf{x})$ is a vertex of the mesh $\mathcal{F}$ after the deformation of the feature points $\mathbf{x}$, while $\mathbf{p}_i^0$ is its initial position. d_i is the distance between $\mathbf{p}_i^0$ and $\mathbf{z}_i^0$, the closest point of the sketch $\mathbf{z}$ before any deformation of the mesh. $d_{max} = \max d_i$.

To minimize the energy, a Newton-Raphson scheme is employed. The sketch abstraction can be approximated as:

$$\mathbf{z}(\mathbf{x}) \approx \mathbf{z}(\mathbf{x}_0) + \mathbf{J}_\mathbf{z}(\mathbf{x}_0) \cdot (\mathbf{x} - \mathbf{x}_0) \tag{3.37}$$

where $\mathbf{J}_\mathbf{z} = \frac{\partial \mathbf{z}}{\partial \mathbf{x}}$ is the Jacobian matrix of the sketch abstraction $\mathbf{z}$ with respect to the feature points vector $\mathbf{x}$. For what concerns the iterative step of the Newton's method, the gradient is derived as:

$$\frac{\partial E_s}{\partial \mathbf{x}} = \mathbf{J}_\mathbf{z}^T \frac{\partial E_s}{\partial \mathbf{z}} \tag{3.38}$$

and the Hessian as:

$$\frac{\partial^2 E_s}{\partial \mathbf{x}^2} = \mathbf{J}_\mathbf{z}^T \frac{\partial^2 E_s}{\partial \mathbf{z}^2} \mathbf{J}_\mathbf{z}. \tag{3.39}$$

However, while the regularization term R_p is inexpensive to compute, R_d is included in the iteration step of the Newton's method, as it depends directly

on $\mathbf{p}(\mathbf{x})$. Therefore, 3.38 and 3.39 become, respectively:

$$\frac{\partial E_s}{\partial \mathbf{x}} = \mathbf{J}_{\mathbf{z}}^T \frac{\partial E_s}{\partial \mathbf{z}} + \mathbf{J}_{\mathbf{p}}^T \frac{\partial E_s}{\partial \mathbf{p}} \tag{3.40}$$

and

$$\frac{\partial^2 E_s}{\partial \mathbf{x}^2} = \mathbf{J}_{\mathbf{z}}^T \frac{\partial^2 E_s}{\partial \mathbf{z}^2} \mathbf{J}_{\mathbf{z}} + \mathbf{J}_{\mathbf{p}}^T \frac{\partial^2 E_s}{\partial \mathbf{p}^2} \mathbf{J}_{\mathbf{p}}. \tag{3.41}$$

3.2.3 Implementation

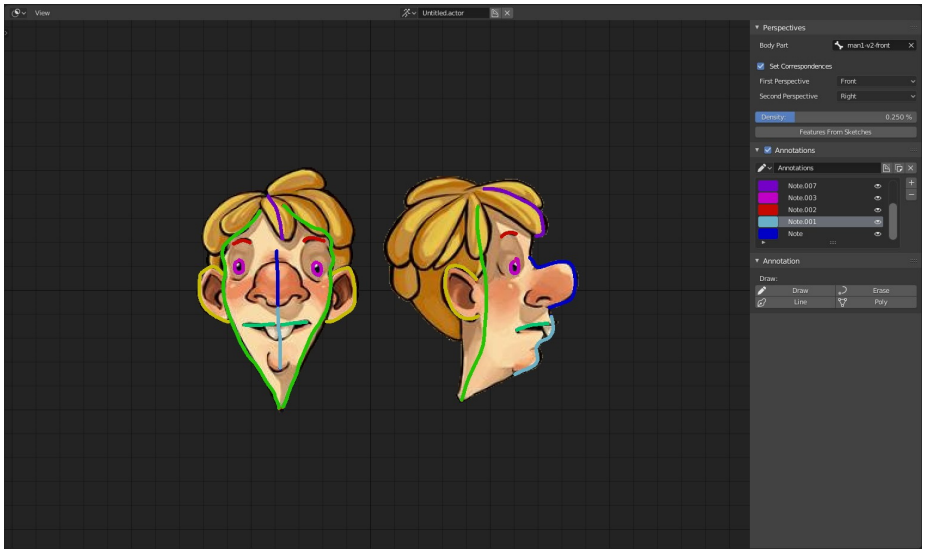


Figure 3.15: The editor in Blender to draw the feature points on the input images (Copyright ©, Caitlyn Patten)

The system has been implemented within Blender. The open source nature of Blender allowed the integration of the system by customizing its tools. One of these tools is the new editor to draw the sketches to define the feature points on top of the input images, which uses Blender's native tool *Grease Pencil*, as shown in figure 3.15. The user can set the required sample rate to generate the feature points from these sketches. In this editor, the user can also draw the sketch for the Silhouette energy 3.34. These sketch will be treated as the others to generate the feature points. During the computation of the energy, however, these sketches will be re-sampled with a higher sample rate, to generate a better shape.

To generate the triangle meshes from the 2D images, the contour of the images is first computed using the OpenCV library [Bradski, 2000]. Then the contour is sampled and the mesh is generated as a constrained conforming Delaunay triangulation using the Triangle library [Shewchuk, 1996]. This

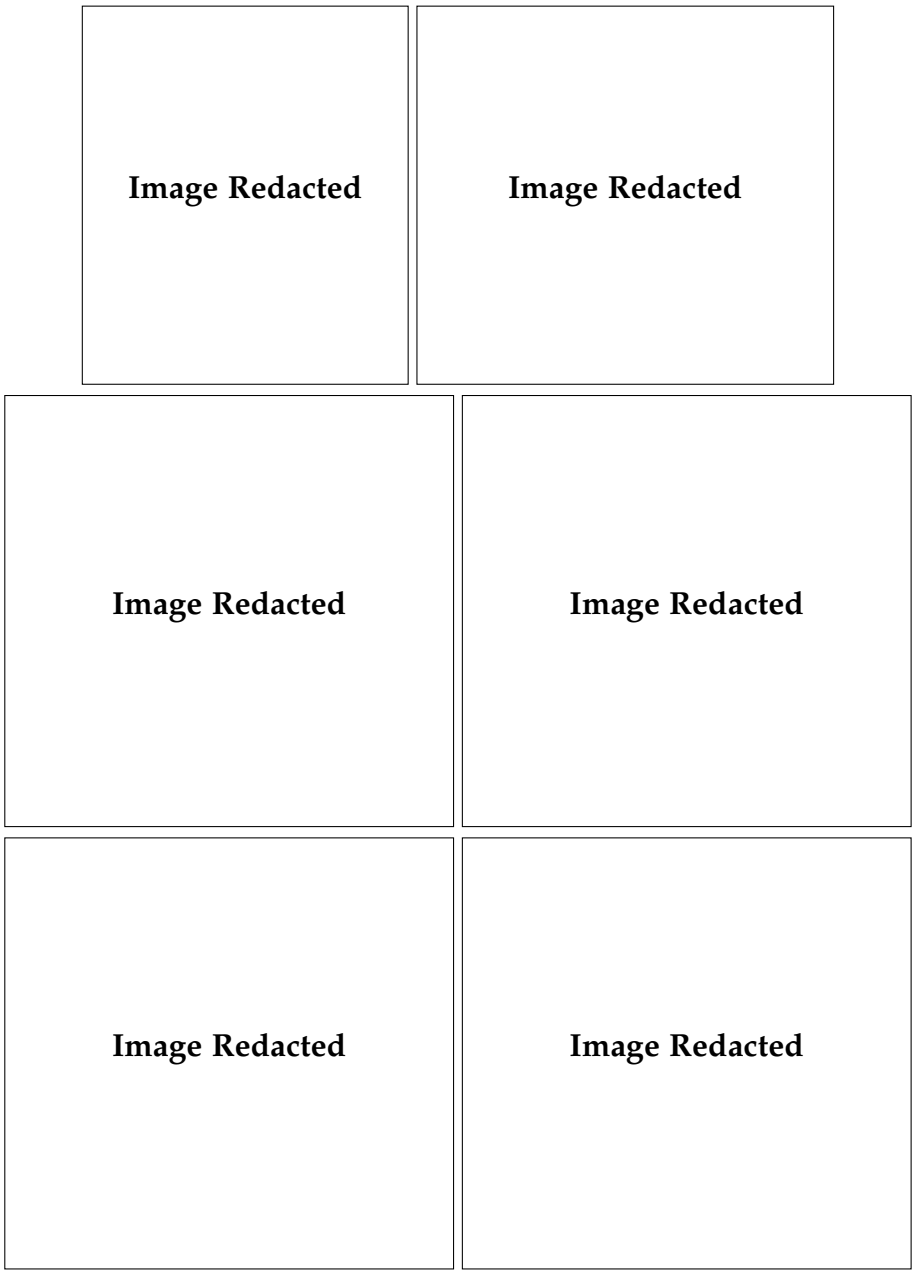


Figure 3.16: One of the models generated from the images in the first row.

library not only allows the inclusion of constraint points, but also the enforcement of specific edges in the triangulation, which works synergistically with the feature points marked with sketches. In fact, the deformation of the mesh when the feature points are connected directly with edges helps maintaining specific shapes intact, such as the nose or the eyes. For what concerns the

distance transform, it has been computed with OpenCV library.

The energy is computed using Matlab, but it has been included in Blender using Matlab Runtime. The system uses Matlab own solver with the BFGS Quasi-Newton algorithm. The 3D model is generated and shown in Blender's 3D Viewer. Figure 3.16 shows one of the models generated with the system. While the modelled character maintains the similarity with the original drawing of the side view, the back part of the head is missing as the front view, from which has been generated, does not include that section in the original drawing. It could be however included by modelling the back part and merging the two models. Section 3.4 will discuss the obtained results and time to solve the minimization problem.

3.3 3D Animation from Sketches

The work presented in this section of the thesis has been originally presented as an independent tool for posing 3D characters with a sketch-based interface. However, it has been used, for the purpose of this thesis and in the pipeline described in chapter 4, to automatically generate 3D animations from 2D one. In fact, the 2D skeleton of the original 2D animation can be assimilated to a 2D sketch, which is then used as input of this method. The work described in this section has been published as a poster at SIGGRAPH 2016 [Barbieri et al., 2016].

In the 2D production, drawings have been used for decades as a design method, since they are an intuitive interface and allow the users to draft fast prototypes of their ideas. In contrast, for 3D productions, it is required a deep knowledge of specific tools and complex notions. It is clear that there is a significant imbalance of skill required to work with a 2D or a 3D production. In an attempt to simplify the 3D productions, researchers are trying to bring the sketching interfaces in almost every computer graphics tasks.

In the current 3D animation pipeline, hand-drawn sketches are already employed for storyboards during the pre-production phase. However, they are only the instructions for the animators, who have to pose the characters manually following the orders enclosed in them. This process could lead to inaccuracies or misunderstanding of the drawing, so sometimes the animator's work must be redone, or the director has to change the entire frame.

The aim of this method is to use drawings, such as the ones in the storyboards, to pose characters automatically, in an attempt to bring the intuitiveness of the traditional 2D animation in the 3D production pipeline. Many researchers have already tried to propose an approach to solve this problem.

Using sketches in the production phase, through integration with the preproduction phase, would ensue in faster results and a reduction of the costs.

Posing 3D characters through 2D sketches, however, is a complex and open problem. In particular, two different sub-problems have been identified: find the correspondences between the model and the drawing and compute the depth, which is absent in a drawing. In literature, this problem is solved with three different main approaches which have been described in section 2.3.

Inspired by the work of Hahn et al. [2015], a new method to pose character with the sketch of the skeleton or the outline of the character is introduced in this section. Given a rigged 3D mesh $\mathcal{M}$ with vertices $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n)$, the rig parameters are defined as p, and the deformation of the vertices by the rig parameters is described as a linear blend skinning:

$$x_i(p) = \sum_{j=1}^{q} \mathbf{W}_{i,j} \cdot \mathbf{T}_j(p) \cdot x_i^0 \tag{3.42}$$

where q is the number of joints, $\mathbf{W}$ is a matrix containing skinning weights, $\mathbf{T}_j$ is the transformation matrix of the j-th joint and x_i^0 is the i-th vertex in the original surface.

The method takes a sketch $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_m)$ as input, and samples the points of the skeleton or the outline of the character, based on the camera view set while the user took the drawing, as a set of points $\mathbf{z} = (\mathbf{z}_1, \mathbf{z}_2, \cdots, \mathbf{z}_l)$. The method compute both the skeleton and the outline, however they are computed and used separately, to avoid mismatch of the points.

The points of $\mathbf{z}$ are represented as a function of p, so that p can be used as a parameter of the energy function. The points of the set $\mathbf{z}$ are projected on the surface of the mesh, and the system picks the three closest vertices $\mathbf{p}_a^i$, $\mathbf{p}_b^i$ and $\mathbf{p}_c^i$. The system then computes the barycentric coordinates α_i, β_i and γ_i for the point $\mathbf{z}_i$ with respect to the triangle composed by the three points picked previously. The point $\mathbf{z}_i$ is expressed in function of the feature points p as:

$$\mathbf{z}_i(p) = \alpha_i \cdot \mathbf{x}_a^i(p) + \beta_i \cdot \mathbf{x}_b^i(p) + \gamma_i \cdot \mathbf{x}_c^i(p) \tag{3.43}$$

The objective is to match the points of the two sets $\mathbf{z}$ and $\mathbf{y}$, and meanwhile to deform $\mathbf{z}$ to $\mathbf{y}$ as closely as possible. Inspired by [Chui and Rangarajan, 2003], the *binary linear assignment-least squares* energy function is defined as:

$$\min_{\Omega,p} \sum_{i=1}^{m} \sum_{j=1}^{l} \omega_{ij} \|\mathbf{y}_i - \mathbf{z}_j(p)\|^2 + \Phi(p) - \zeta \sum_{i=1}^{m} \sum_{j=1}^{l} \omega_{ij} \tag{3.44}$$

subject to:

$$\sum_{i=1}^{m+1} \omega_{ij} = 1, \quad j \in \{1, 2, \ldots, l\}$$
$$\sum_{j=1}^{l+1} \omega_{ij} = 1, \quad i \in \{1, 2, \ldots, m\}$$
$$\omega_{ki} \in \{0, 1\}$$

$\Phi(p)$ is a regularization term defined as follow:

$$\begin{aligned} \Phi(p) = \lambda_{pose} \|p - p_0\|^2 + \\ + \lambda_{plane} \sum_{i=1}^{m} (\mathbf{v}^T (\mathbf{x}_i(p) - \mathbf{x}_i^0)) + \\ + \lambda_{dist} \sum_{i=1}^{m} \frac{d_i}{d_{max}} \|\mathbf{x}_i(p) - \mathbf{x}_i^0\|^2 \end{aligned} \tag{3.45}$$

in which λ_{pose}, λ_{plane} and λ_{dist} are weights for each part of the regularization term, $\mathbf{v}$ is the viewing direction of the camera, d_i is the distance, for each vertex $\mathbf{x}_i(p)$, from its position in the undeformed mesh, and d_{max} is the maximum d_i. The first component of the regularization term is needed to avoid solutions too far from the original pose. The second component is used in order to try to keep the deformations of the 3D surface points inside their viewing plane. Finally, the third component penalises deformations of body parts too far away from the sketches.

The matrix $\Omega = \{\omega_{ij}\}_{(m+1)\times(l+1)}$ is the binary correspondence between the sets of points $\mathbf{z}$ and $\mathbf{y}$, and it consists of two parts. The inner $m \times l$ part defines the correspondences. If a point $\mathbf{z}_j$ corresponds to a point $\mathbf{y}_i$, then $\omega_{ij} = 1$, otherwise $\omega_{ij} = 0$. The row and column summation constraints guarantee that the correspondence is one-to-one. The extra $m + 1$ row and $l + 1$ column are introduced to handle the outliers (that also include the points having no correspondence). Once a point is rejected as an outlier, the extra entries will start taking non-zero values to satisfy the constraints, when the related inner entries all becomes zero. ζ is a parameter used to prevent the rejection of too many points as outliers.

The energy function 3.44 consists of two interlocking optimisation problems: a linear assignment discrete problem on the correspondence and a non-linear continuous problem for the deformation. This approach, however, is not meaningful at each step, in particular when the deformation is far away from the optimal solution. For this reason two techniques have been adopted: *softassign* and *deterministic annealing*. The idea of the softassign is to relax the

binary correspondence variable Ω to be a continuous valued matrix $\mathbf{W}$ in the interval $[0, 1]$ while enforcing the row and column constraints.

The original binary assignment-least squares problem 3.44 is converted to the following *fuzzy assignment-least squares* energy function:

$$\min_{\mathbf{W},p} \sum_{i=1}^{m}\sum_{j=1}^{l} \mathbf{w}_{ij}\|\mathbf{y}_i - \mathbf{z}_j(p)\|^2 + \Phi(p) + T\sum_{i=1}^{m}\sum_{j=1}^{l} \mathbf{w}_{ij}\log \mathbf{w}_{ij} - \zeta\sum_{i=1}^{m}\sum_{j=1}^{l} \mathbf{w}_{ij} \tag{3.46}$$

subject to:

$$\begin{aligned}
&\sum_{i=1}^{m+1} \mathbf{w}_{ij} = 1, \quad j \in \{1, 2, \ldots, l\} \\
&\sum_{j=1}^{l+1} \mathbf{w}_{ij} = 1, \quad i \in \{1, 2, \ldots, m\} \\
&\mathbf{w}_{ij} \in [0, 1]
\end{aligned}$$

where T is called the temperature parameter in annealing. When the temperature T reaches zero, the fuzzy correspondence $\mathbf{W}$ becomes binary, and the energy function 3.46 degenerates to 3.44.

The energy function 3.46 can be minimised by an alternating optimisation algorithm that successively updates the parameter $\mathbf{W}$ and the rig parameter p while gradually reducing the temperature T. The minimization problem is solved with the Robust Point Matching (RPM) algorithm, which consists of three steps.

Step 1: Update the Correspondence To update the correspondence, for the points $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, l$:

$$\mathbf{w}_{ij} = \frac{1}{T}\exp\left(-\frac{(\mathbf{y}_i - \mathbf{z}_j(p))^T(\mathbf{y}_i - \mathbf{z}_j(p))}{2T}\right), \tag{3.47}$$

for the outlier entries $i = 1, 2, \ldots, m$ and $j = l + 1$:

$$\mathbf{w}_{i,l+1} = \frac{1}{T_0}\exp\left(-\frac{(\mathbf{y}_i - \mathbf{z}_{l+1}(p))^T(\mathbf{y}_i - \mathbf{z}_{l+1}(p))}{2T_0}\right), \tag{3.48}$$

for the outlier entries $i = m + 1$ and $j = 1, 2, \ldots, l$:

$$\mathbf{w}_{m+1,j} = \frac{1}{T_0}\exp\left(-\frac{(\mathbf{y}_{m+1} - \mathbf{z}_j(p))^T(\mathbf{y}_{m+1} - \mathbf{z}_j(p))}{2T_0}\right), \tag{3.49}$$

where $\mathbf{z}_{l+1}$ and $\mathbf{y}_{m+1}$ are the outlier cluster centres. The rows and columns of $\mathbf{W}$ must be then normalized:

$$\mathbf{w}_{ij} = \frac{\mathbf{w}_{ij}}{\sum_{k=1}^{l+1} \mathbf{w}_{ik}}, i = 1, 2, \ldots, m, \tag{3.50}$$

$$\mathbf{w}_{ij} = \frac{\mathbf{w}_{ij}}{\sum_{k=1}^{m+1} \mathbf{w}_{kj}}, j = 1, 2, \ldots, l. \tag{3.51}$$

Step 2: Update the rig parameters For what concerns the rig parameter, the following optimisation problem is solved:

$$\min_{p} \sum_{i=1}^{m} \sum_{j=1}^{l} \mathbf{w}_{ij} \|\mathbf{y}_i - \mathbf{z}_j(p)\|^2 + \Phi(p) \tag{3.52}$$

which can be solved with a Newton-Raphson scheme. The solution of this similar to the solution of the Silhouette energy solved in section 3.2.2.

Step 3: Annealing The annealing scheme controls the dual process update. Starting at $T_{\text{init}} = T_0$, the temperature parameter T is gradually reduced according to a linear annealing schedule: $T^{\text{new}} = T^{\text{old}} \cdot r$. The parameter T_0 is set to the largest square distance of all point pairs, while r is the annealing rate. r is usually set to 0.93, and normally has a value in the range $[0.9, 0.99]$. Finally, T_{final} is chosen to be equal to the average of the squared distance between the points which are being deformed.

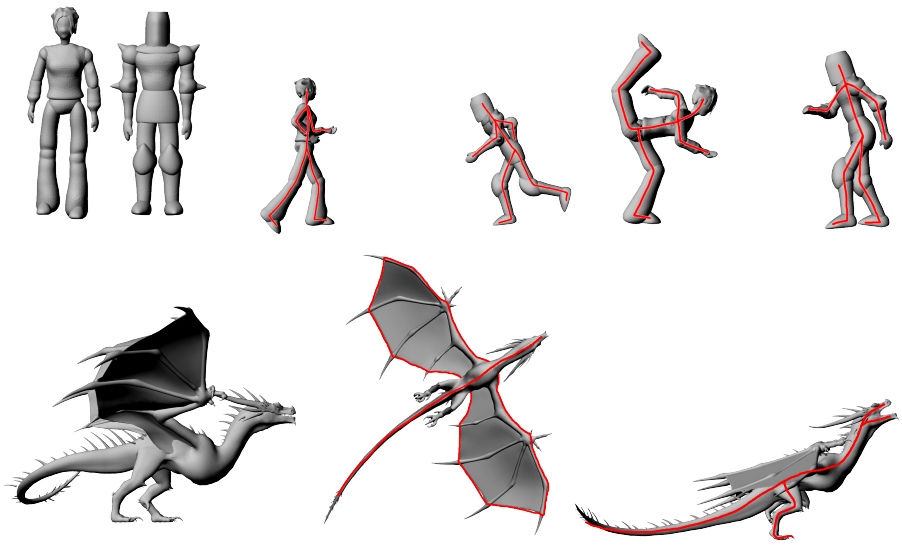


Figure 3.17: Two poses of two characters generated with the tool. On the left are the characters in resting pose. The red lines are the sketches drawn by the user.

A method for sketch-based character posing has been proposed, which is more flexible than state of the art methods proposed in previous years. Compared to [Hahn et al., 2015], a way to compute automatically a set of points analogue to the sketch abstraction has been introduced. To support this new way to generate the sketch abstraction, the techniques of *softassign* and *deterministic annealing* have also been introduced, together with the RPM algorithm to minimize the energy function, which is proven in the original paper [Chui and Rangarajan, 2003] to be less incline to find local minima and does not rely on heuristics to handle outliers, compared with the ICP method employed by [Hahn et al., 2015]. Figure 3.17 shows some of the results obtained with the tool.

3.4 Analysis of the results

2.5D Modelling The first results analysed in this section are those generated with the 2.5D method, from section 3.1. Although 2.5D have many applications, 2.5D modelling techniques are not exactly popular. Among the works regarding this topic, we believe that [Rivers et al., 2010] is still the state of the art.

Figure 3.18: The results from [Rivers et al., 2010].

The main difference between River's work and the one proposed in this thesis, is that the former provide a tool to draw the character from different perspectives, and it generate the 2.5D model on the fly. It is not possible to use River's method with existing images. The method proposed, instead, it is focused on that, as the main focus is to integrate the tool in an animation pipeline and to generate the 2.5D model from an existing turnaround of a character.

Of course, there are advantages and disadvantages in either cases. River's method, in particular, is flexible on the viewing angle, as it allows the user to input new drawings from any direction. This is not the case with the proposed method, which, constrained by the input turnaround, can allow the user to

rotate the character only around the y axis. On the other hand, the proposed method can be integrated in any existing project, as the only requirement is the turnaround of the character. River's method would require to draw again every single character from scratch.

However, there are a few limitation with this method. One of them concerns the identification of the joint part and the margins of each body part, which must be manually set by users and it can be hardly automated, considering cartoon characters can have any kind of shape. A way to do it could be through the optical flow and structure from motion methods, but these kind of systems require accuracy, which is available in photos and videos, but not in drawings. Moreover, cartoons often use uniform colour for entire surfaces, which would result in a lack of details to find feature points automatically, and non consistent contours, which would lead to wrong correspondences.

Second, as the character's parts are still 2D, it must be studied a way to make it interact with its surrounding environment, including any object in the scene. Alternatively, the system could be extended to support any kind of object. Finally, as the method needs to interpolate the mesh between two states, it supports the movement of the camera only horizontally. The vertical movement would require a trilinear interpolation including the top view as well, which, however, is not as common as the turnaround to be produced in the animation pipeline.

For what concerns the results, taking in exam figure 3.9, the interpolation between the two *joint parts* from the two perspectives might results in a deformation of the mesh, that may appear slanted or squeezed. In the figure, this can be observed with the head, that is slanted, and with the torso, that is squeezed. This happens for the different size of the meshes from the two perspectives and the position of some of the elements. All these problems, with the exception of the identification of feature points in the cartoon characters, are solved with the 3D methods presented in this thesis.

3D Modelling from 2D images Two methods of 3D modelling from 2D images have been presented. The first one, presented in section 3.2.1, inflates the mesh based on the distance of the vertices from the contour. This method is supported by a surface registration technique, which balances the missing information in the 3D mesh generation by transforming the model while the camera rotates around the character. A 3D animation is then generated from the 2D one, using the method introduced in section 3.3. The results of the three methods can be seen in figure 3.19.

This method might generates some artifacts during the transformation between two perspectives, as can be noticed in figure 3.20. Moreover, when the

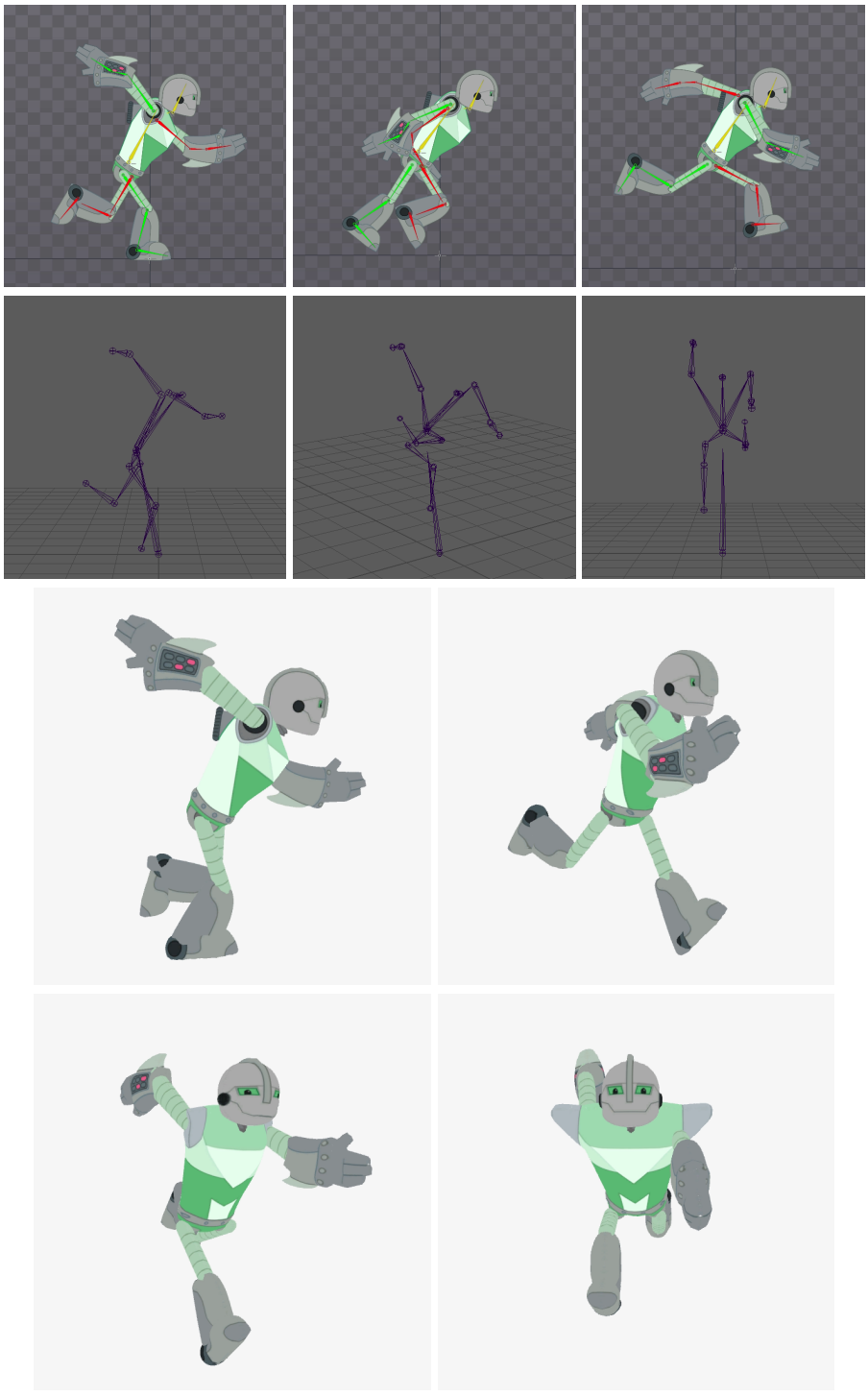


Figure 3.19: The first row shows the original 2D animation. The second row shows a frame of the generated 3D animation. The two final rows show the result obtained by the inflated 3D model, combined with the registration method and the generated 3D animation. (Copyright ©, Thud Media)

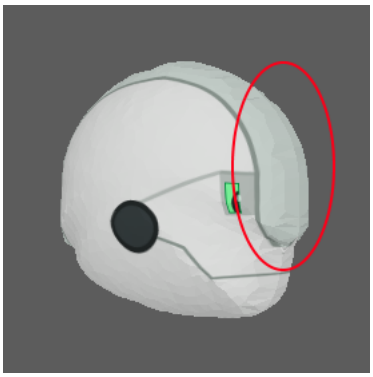


Figure 3.20: Some artifacts generated by the system during the transformation between two perspectives. (Copyright ©, Thud Media)

texture on the mesh is switched from one perspective to the next, the transition might result too sudden, in certain cases. Both problems could be solved adding more perspectives to the system. The example in the figure has been obtained by a turnaround of four perspectives. In-between images would require more models to generate, but the result would be smoother.

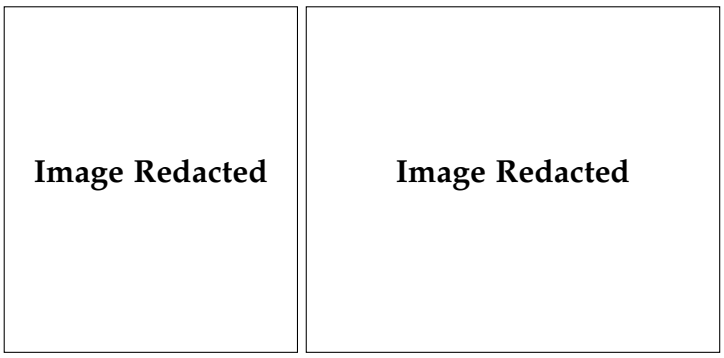


Figure 3.21: The hair in front of the face cannot be generated by the method.

The second method for the generation of the 3D model from 2D images solves an optimization problem, with equations that include information from the side views, in order to obtain a 3D model which can be used without the support of techniques such as the surface registration.

This methods can generate approximations of cartoon characters faces from just drawings, however there are a few limitations. First of all, character with prominent features in front of the face cannot be generated. An example is shown in figure 3.21. An alternative solution might be, if possible, to separate that feature from the main image of the face, generate the two shapes separately, and then merge them together again. Another limitation is that the modelling requires some user input. The feature points expressed with

Character	Figure	Vertices	Time (s)
Boy 1	3.24	928	5.644
Boy 2	3.25	1200	7.176
Girl 1	3.26	1213	9.661
Girl 2	3.27	2022	16.581
Man 1	3.28	450	5.256
Man 2	3.29	639	7.965
Man 3	3.30	977	6.874
Robot	3.31	237	3.257
Woman	3.32	180	1.378

Table 3.1: This table shows the number of the vertices of the mesh, and the time it takes to solve the optimization problem and generate the 3D model.

sketching are more intuitive from the user's perspective, however the modelling inevitably depends on the quality of the input sketches. As mentioned for the 2.5D modelling, optical flow and structure from motion methods could be employed in this case too to automatically compute the feature points, but the same consideration made in that instance apply here as well.

The images between 3.24 and 3.32 show the difference between the model generated with inflation and the one generated with the optimization method, and table 3.1 shows the time it takes to generate them.

3.5 Evaluation of the methods

For what concerns the evaluation of the methods presented in this chapter, two factors have been considered: the time to generate the model and the quality of the model.

The first factor, time, is the most important. As mentioned earlier, handmade drawings are not accurate; therefore, the introduced method only generate an approximation of the 3D character. Hence it is crucial for the methods to be beneficial in a production environment, that the time required to generate the approximated character, combined with the time to refine it, is lower than the time necessary to model the character from scratch. The second factor, quality, has a consequence on the first. In fact, a better generated model will require less refining time for the artist. These two factors have been used for the evaluation of both the 3D modelling and 3D animation method introduced in this chapter.

For the evaluation of the modelling, a test has been carried out with five artists from Cloth Cat Animation. To test the time, they have been asked to model the nine characters used for the experiments of the optimization method of section 3.2.2 — in the figures between 3.24 and 3.32.

Character	Figure	Artists time	System time	Quality (1-7)
Boy 1	3.24	50 m	6 m 17 s	4.6
Boy 2	3.25	41 m	5 m 30 s	4.4
Girl 1	3.26	38 m	5 m 06 s	4.6
Girl 2	3.27	45 m	4 m 22 s	4.6
Man 1	3.28	58 m	5 m 31 s	4.4
Man 2	3.29	36 m	4 m 57 s	3.4
Man 3	3.30	44 m	5 m 43 s	5.2
Robot	3.31	35 m	6 m 46 s	5.8
Woman	3.32	55 m	4 m 18 s	2.8

Table 3.2: A comparison of the average time it took for some of the artists from Cloth Cat Animation to manually model the characters and how long it took to generate them with the methods presented in section 3.2. The last column is an evalutation of the generated models, compared to the manual.

To test the quality, they have been asked to judge the same character modelled with the mentioned methods. The evaluation has been carried out with a single question, asking them to rate the generated model with a number between 1 and 7, where a higher rating meant a better result. Specifically, 1 meant that the artist would need to remake the model from scratch, while 7 that the model did not need any refinement.

Table 3.2 shows the result of the analysis made with the artists. The column *Artist time* shows the average time required by the artists to model the character from the ground up, while the column *System time* shows the time required with the methods presented in this thesis, including the time to set up the points and generate the models. Figure 3.22 displays the comparison of the time in a graphical way. This figure also includes the minimum and maximum time it took the artists to model a specific character.

This comparison highlights the significant difference between the modelling of the characters from scratch and their generation with the methods presented in this thesis. For the time to generate the characters, have also been included the times it has been necessary to adjust the correspondence points and regenerate the models. This entails that the artists would take even less time with this system, depending on the experience on how to set up the correspondence points.

For what concerns the quality of the models, as expected from the limitations of the methods, the models rated worse from the artists are those that have obstacles in front of the face, such as the character in figure 3.32. Conversely, the character of the robot, in figure 3.31, obtained the highest scores, as it is the one with less accessories.

Although none of the models has been judged by the artists ready to be

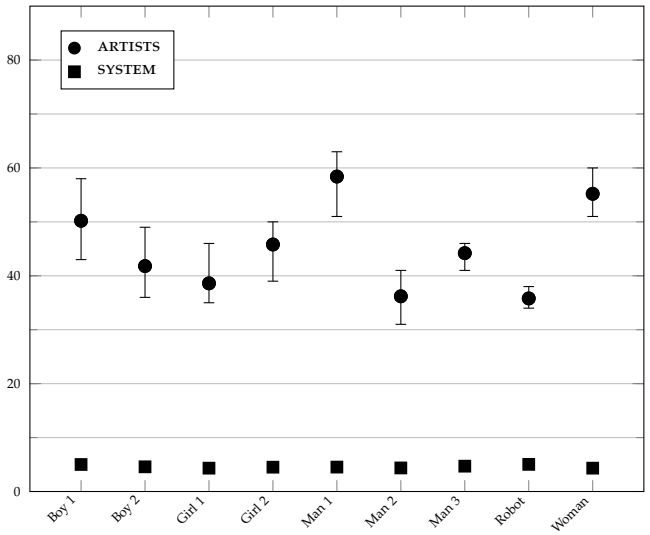


Figure 3.22: A comparison of the average time it took for some of the artists from Cloth Cat Animation to manually model the characters and how long it took to generate them with the methods presented in section 3.2.

used directly after its generation, only two results are below the average. Moreover, the difference in time to obtain the two results is such that it can balance the time for the refinements.

Character	Artists time	System time	Quality (1-7)
Run	56 m	4 m 42 s	6.4
Jump	41 m	4 m 18 s	6.0
Swim	55 m	4 m 48 s	6.2

Table 3.3: A comparison of the average time it took for some of the animators from Cloth Cat Animation to manually animate three animations and how long it took to generate them with the method presented in section 3.3. The last column is an evalutation of the generated animations, compared to the manual.

The same analysis has been carried out with the method for the generation of 3D animations. In this case, five animators from Cloth Cat have been asked to reproduce three 2D animations in 3D. For this case, table 3.3 shows the times for both the animators and the system, and the ratings of the generated animations. Figure 3.23 shows the time comparison in a graphical way.

The animation method does not require any correspondence point, so the times consists only on passing the 2D frames to the system and generate the 3D poses for each keyframe. For what concerns the time difference, the analysis is analogue to the modelling. The time saved using this method to pose the characters is significant for the animations as well. The quality of the anima-

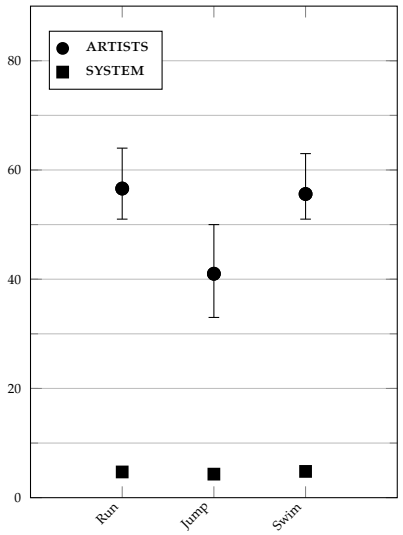


Figure 3.23: A comparison of the average time it took for some of the artists from Cloth Cat Animation to manually animate a character and how long it took to generate the animations with the method presented in section 3.3.

tions, however, has been judged positively by the animators, and all of them have been deemed usable with just few adjustments.

3.6 Conclusion

This chapter introduced three novel methods for employ 2D characters in 3D. The first method presented a unique way to use existing images for the generation of a 2.5D model, exploiting billboarding, parallax scrolling and 2D shape interpolation. The other two methods generate a 3D model, one based on inflation used in combination with a surface registration method, while the second solves an optimization problem and uses information from the side views. The next chapter will introduce an innovative pipeline for animation that will include a game engine. The 3D methods presented in this chapter are integrated in the modelling stage of the pipeline.

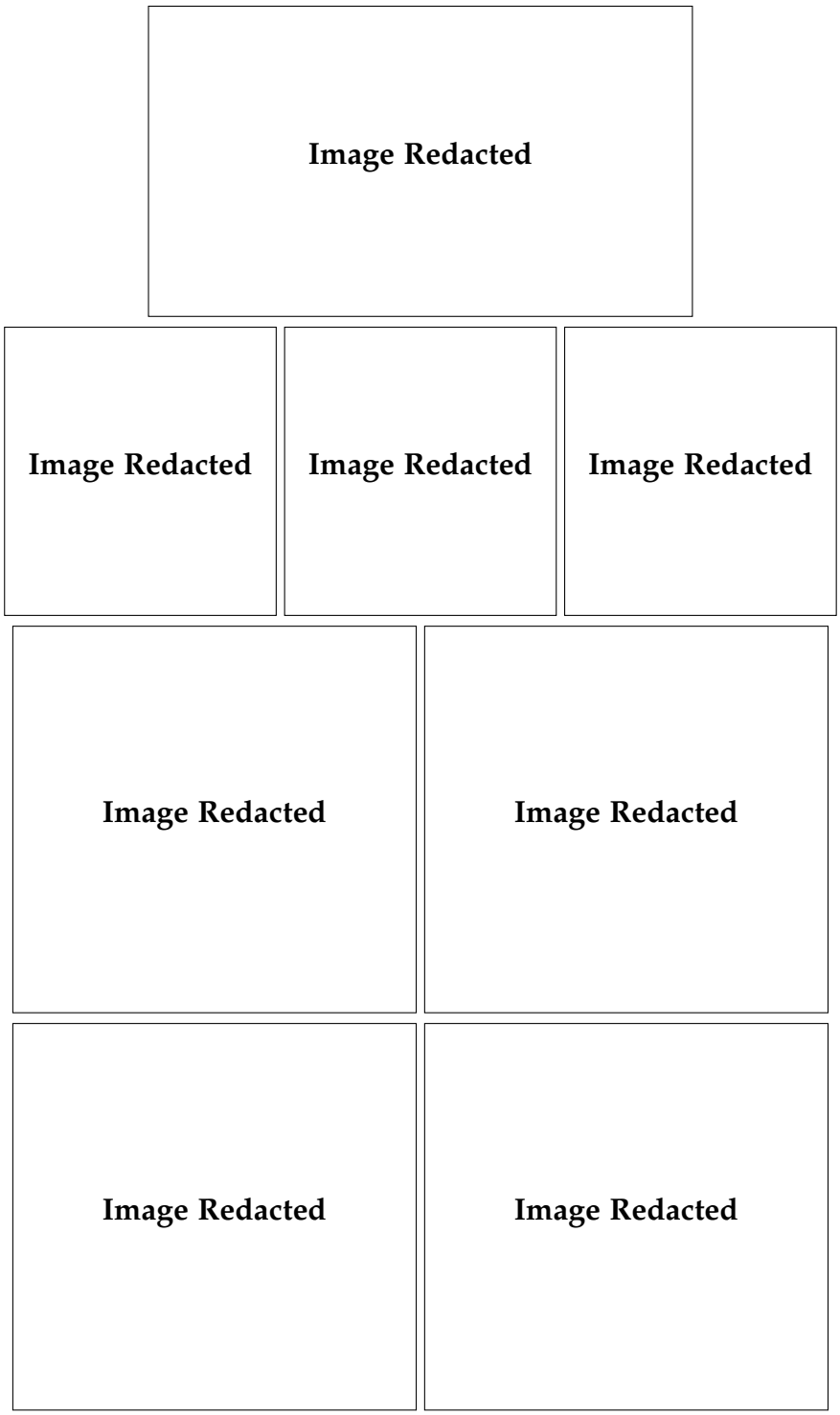


Figure 3.24: Results for Boy 1. The first line shows the correspondences points automatically sampled from the sketches. The second line shows the inflated model. The third and fourth line show the model generated with the optimization method.

Figure 3.25: Results for Boy 2. The first line shows the correspondences points automatically sampled from the sketches. The second line shows the inflated model. The third and fourth line show the model generated with the optimization method. (Copyright ©, Magic Mall, Cloth Cat Animation)

Figure 3.26: Results for Girl 1. The first line shows the correspondences points automatically sampled from the sketches. The second line shows the inflated model. The third and fourth line show the model generated with the optimization method. (Copyright ©, Magic Mall, Cloth Cat Animation)

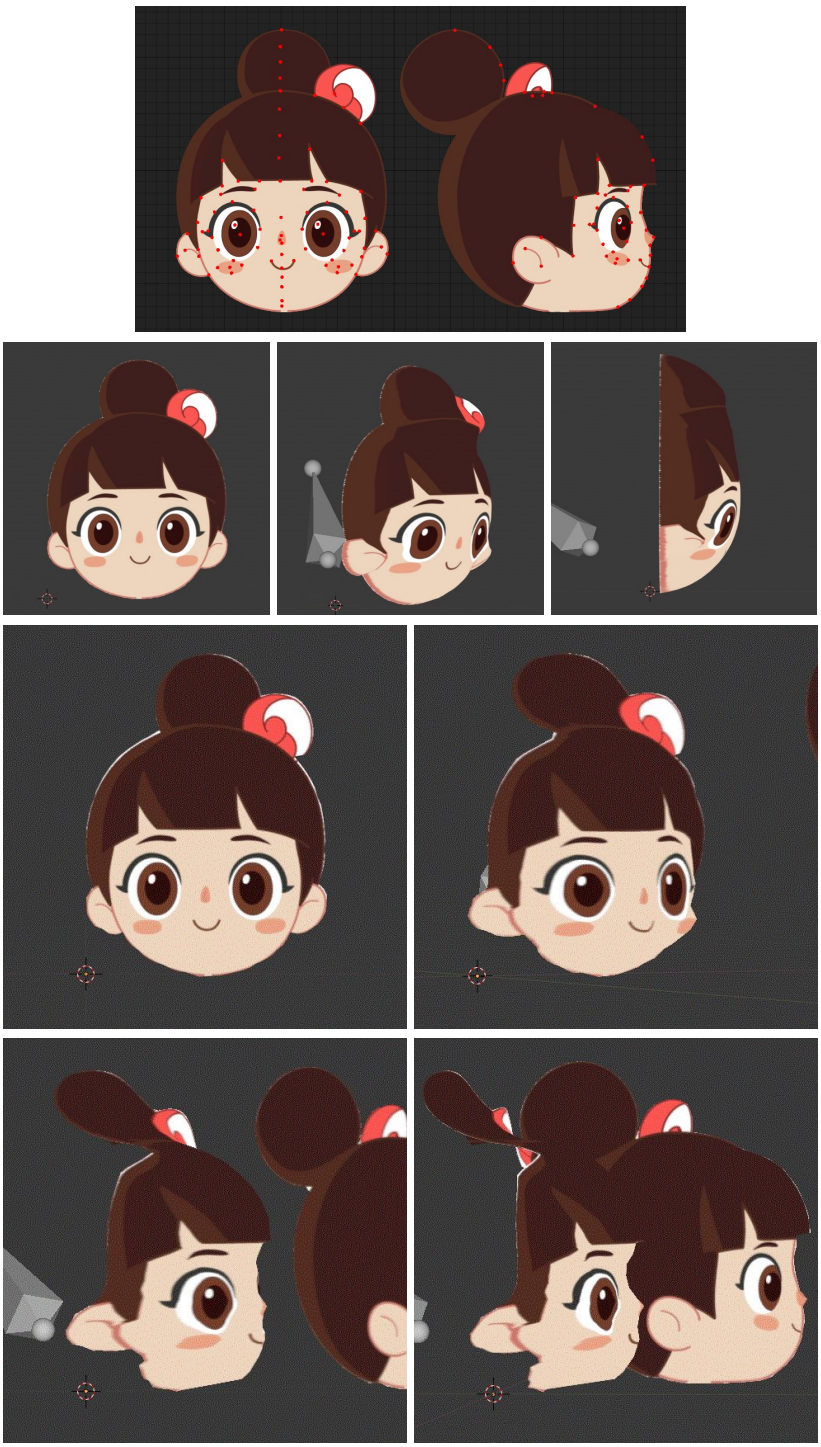


Figure 3.27: Results for Girl 2. The first line shows the correspondences points automatically sampled from the sketches. The second line shows the inflated model. The third and fourth line show the model generated with the optimization method. (Copyright ©, Magic Mall, Cloth Cat Animation)

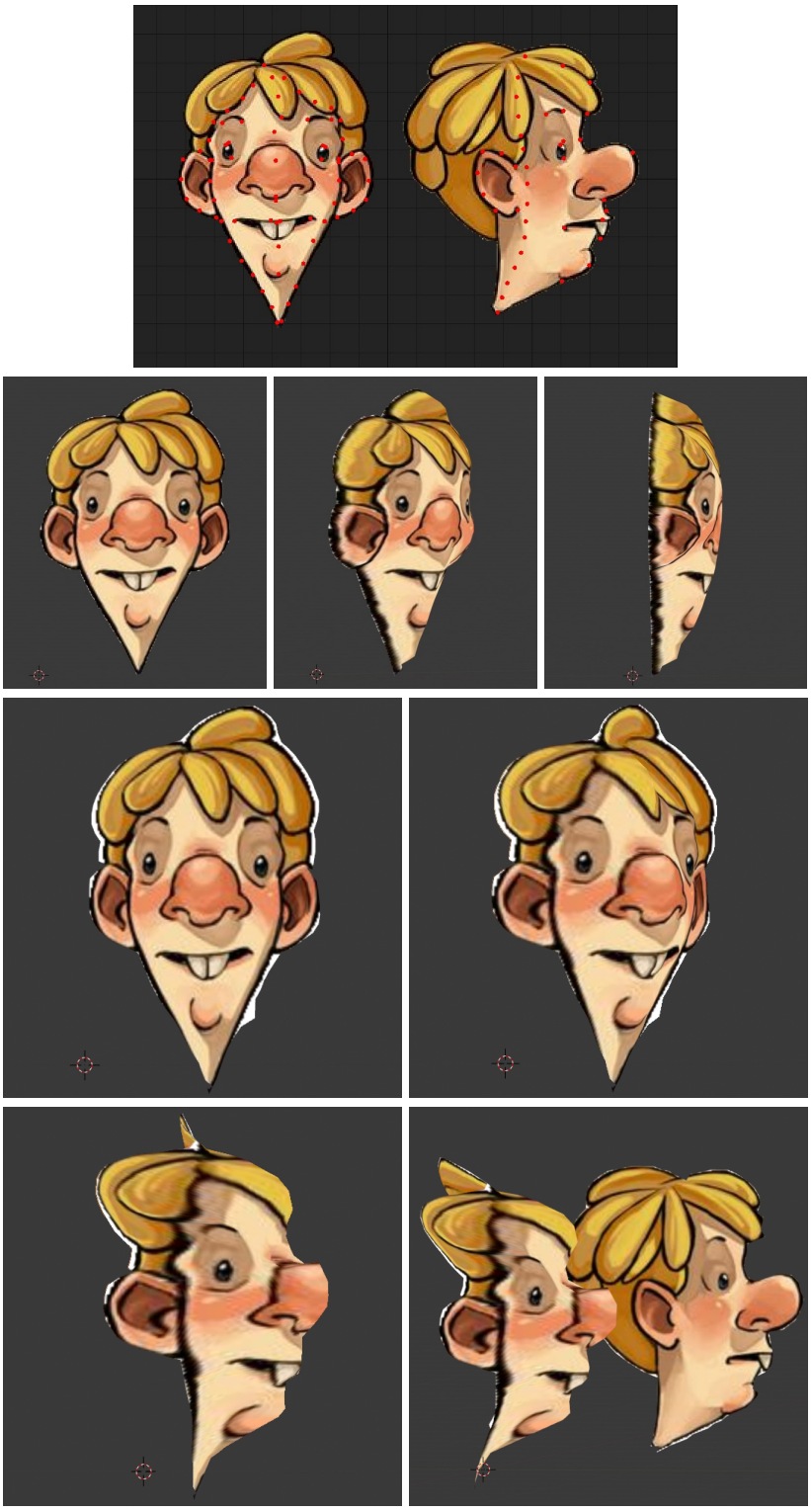


Figure 3.28: Results for Man 1. The first line shows the correspondences points automatically sampled from the sketches. The second line shows the inflated model. The third and fourth line show the model generated with the optimization method. (Copyright ©, Caitlyn Patten)

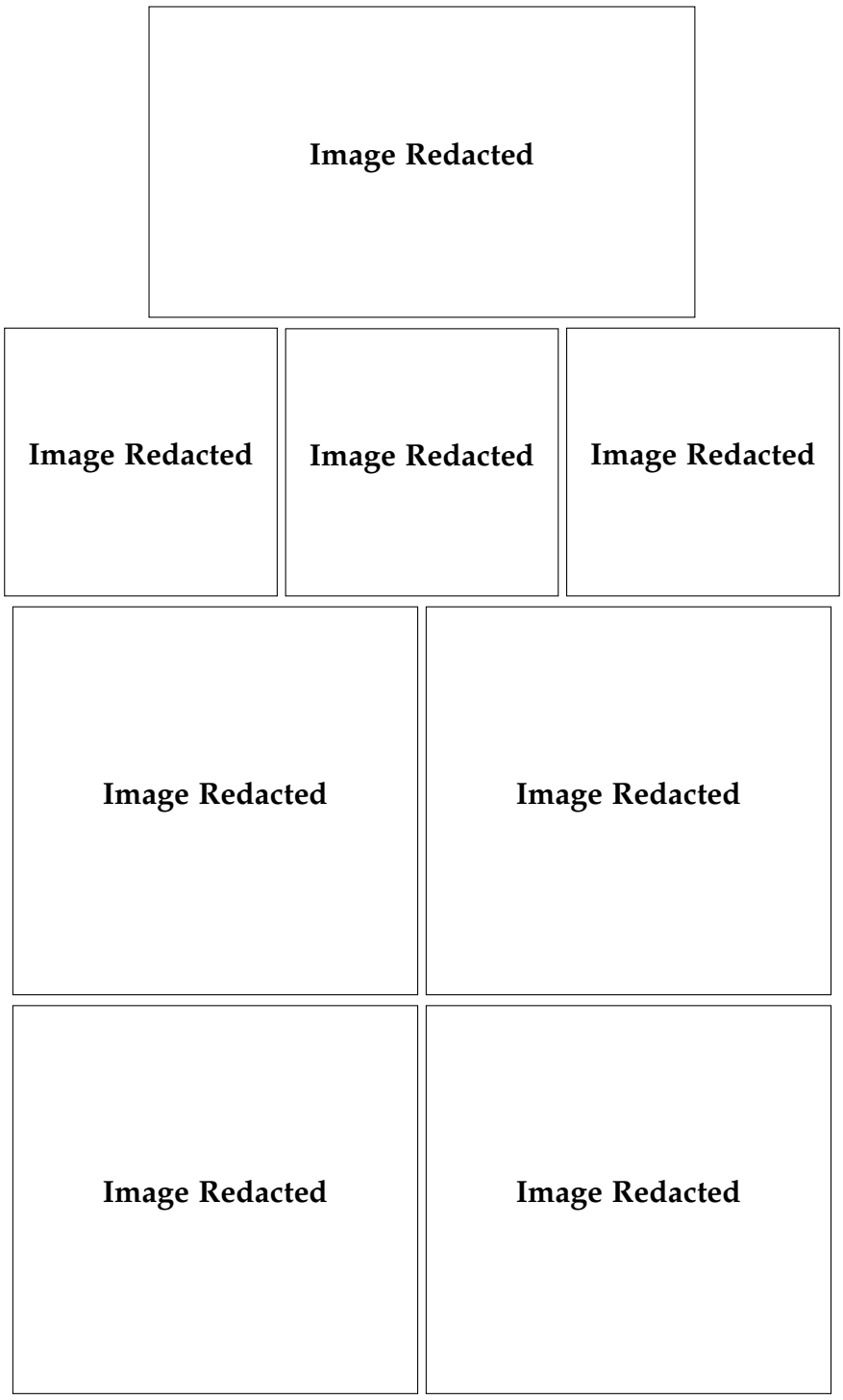


Figure 3.29: Results for Man 2. The first line shows the correspondences points automatically sampled from the sketches. The second line shows the inflated model. The third and fourth line show the model generated with the optimization method.

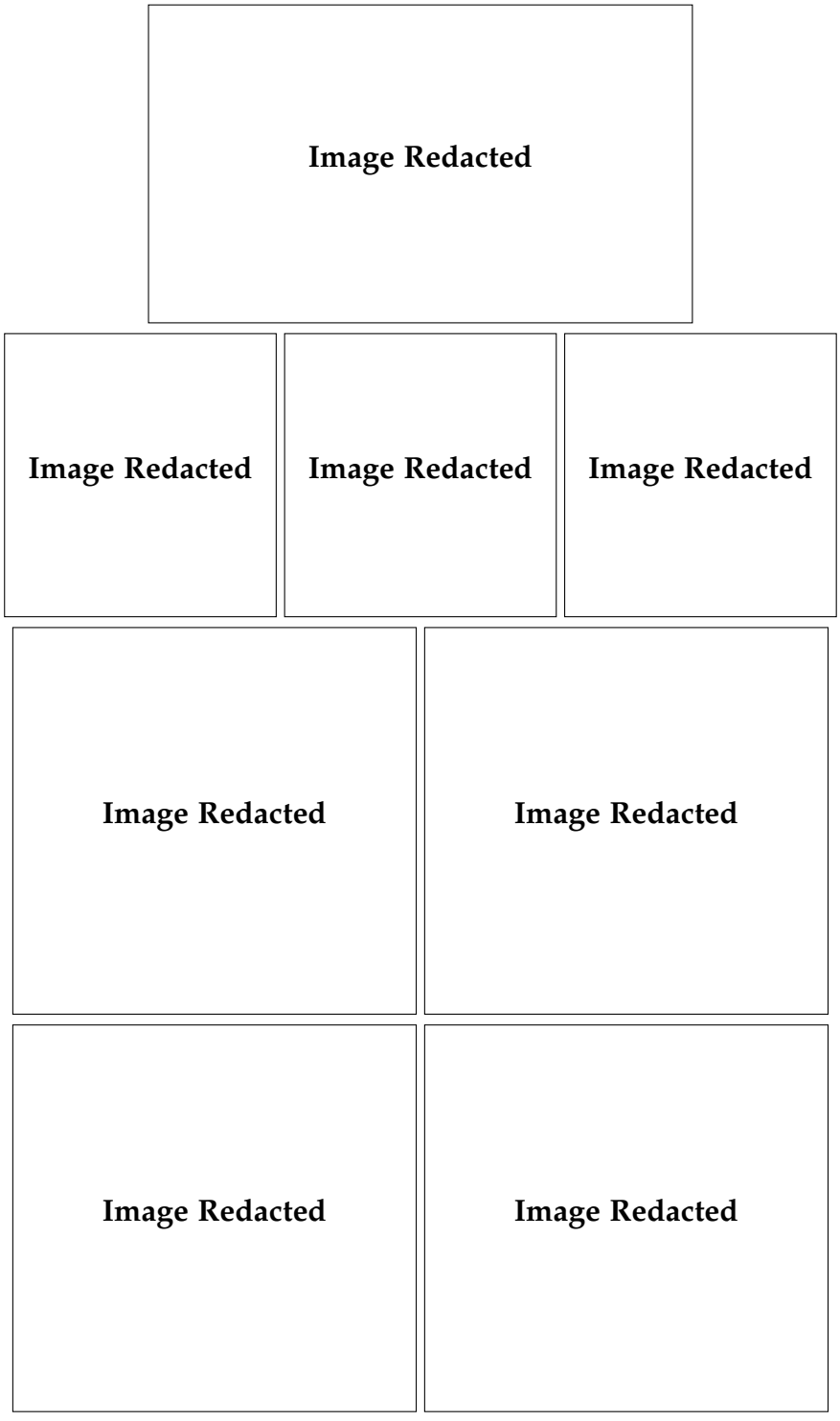


Figure 3.30: Results for Man 3. The first line shows the correspondences points automatically sampled from the sketches. The second line shows the inflated model. The third and fourth line show the model generated with the optimization method.

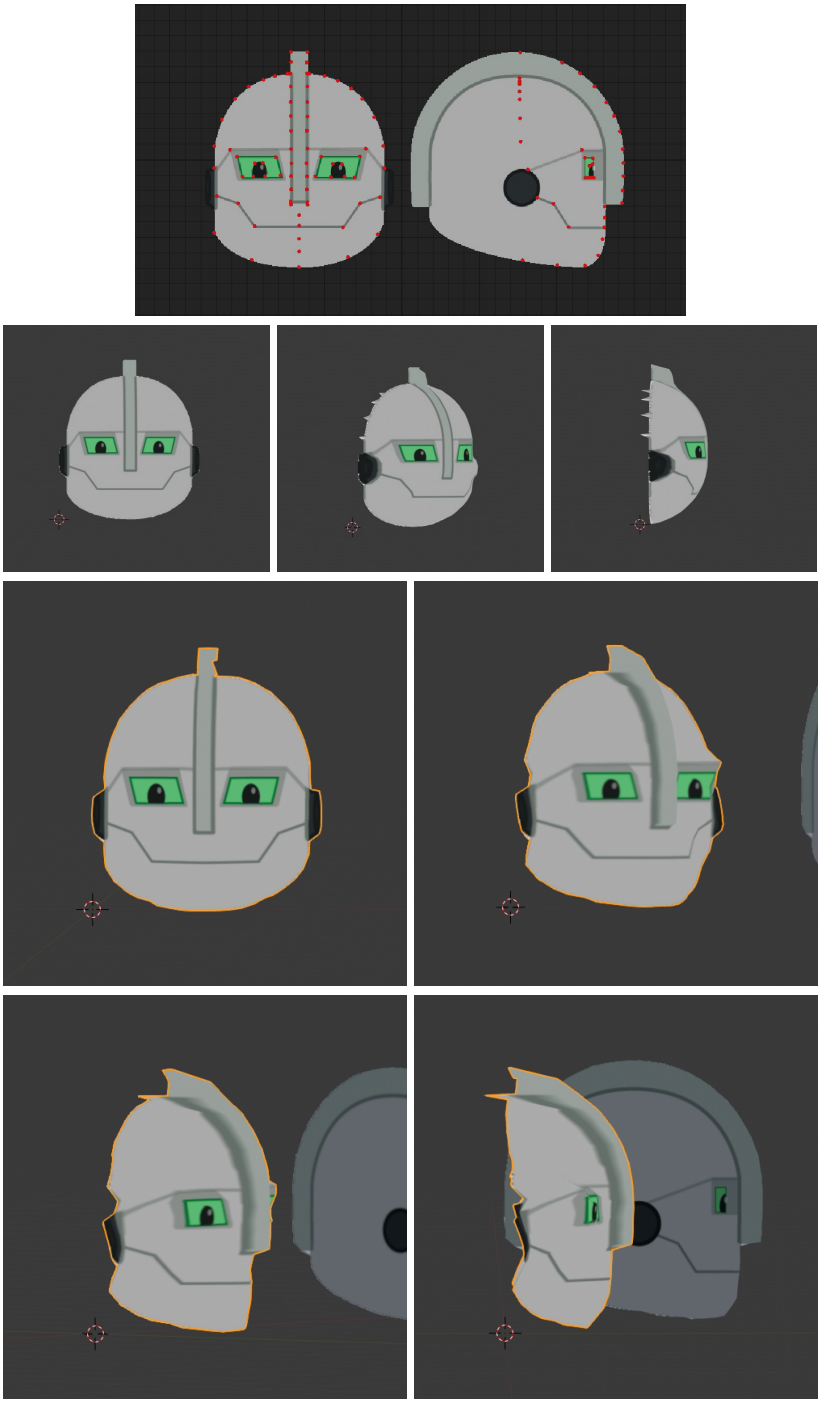


Figure 3.31: Results for Boy 2. The first line shows the correspondences points automatically sampled from the sketches. The second line shows the inflated model. The third and fourth line show the model generated with the optimization method. (Copyright ©, Thud Media)

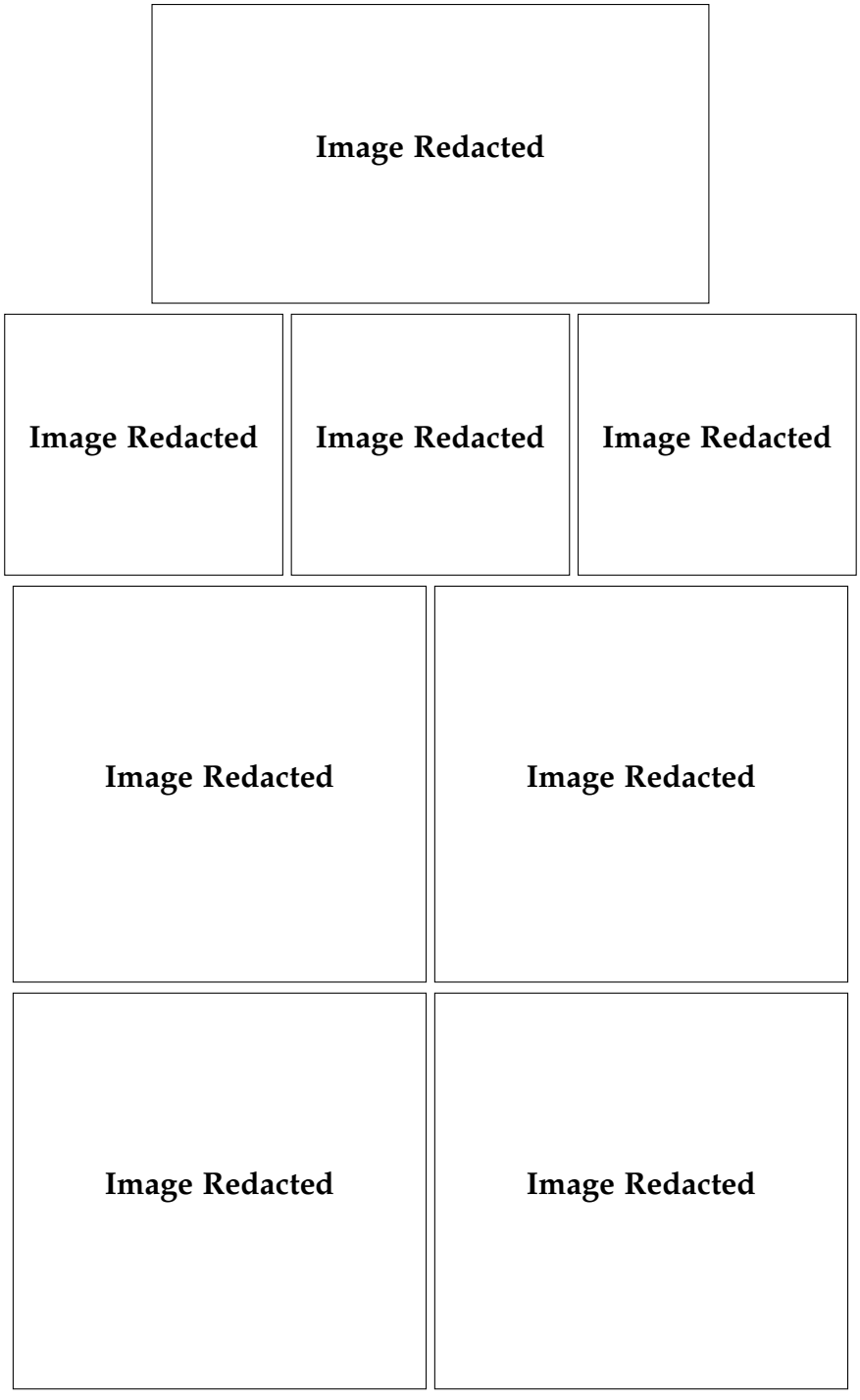


Figure 3.32: Results for Woman. The first line shows the correspondences points automatically sampled from the sketches. The second line shows the inflated model. The third and fourth line show the model generated with the optimization method.

Chapter 4

New Animation Pipeline

The idea of a pipeline involving a game engine is still new in the animation industry. While in section 1.4 some of the works that employed a game engine in their production have been introduced, an actual pipeline involving a game engine has been formally introduced by Epic Games while promoting its trailer for *Fortnite* [Pohl et al., 2018]. Section 2.5.3 offered a comparison of Epic Games' pipeline with what has been tried to achieve with the pipeline presented in this chapter. This chapter introduces a new unified pipeline for animation of films and video games, together with the motivations that brought us to its design and the advantages that it offers. Furthermore, this chapter provides a description step by step of this new pipeline and an analysis of the communication between the two main platforms that compose it.

4.1 Motivation

The employment of game engines in the animation pipeline offers several benefits to the production. One of the main advantages is their capability of real-time rendering. As video games require at least 30 renders per second to be enjoyable, the game engines optimize their rendering process to achieve that goal. An immediate preview of the animation with a quality level comparable with the final product assists the animators in producing a better results in a minor time. Additionally, a faster and accurate preview of the animation opens to more creativity during production, as it makes possible to test ideas that with long rendering time would have been possible to try only with an inaccurate preview.

Animators can get direct accurate visual feedback as they work, and instead of getting a *playblast* — a quick, low quality preview — from Maya, they are able to see their animation in the scene fully rendered. This will allow

animators to early identify errors such as surface intersections immediately. Surface intersection errors have been a serious issue in the past series from Cloth Cat, which caused a considerable loss of time during production. In fact, it resulted hard for the animators to identify intersections even with rigs coloured with low resolution texture maps. Often a shot would be animated, approved, lit and rendered, before an intersection would be identified by the editor reviewing the sequence. This would result in going back to animation, lighting, rendering and another animation check. Sometimes the rig might have needed changes, resulting in going further back in the pipeline. That could sometimes take a day or more in each department. With the build up of shots and fixes in each department, it could have been a few weeks before the sequence was fully rendered again. With the previous system, scenes were so heavy that animators needed to work with proxy meshes, meaning that instead of animators seeing a row of bushes or trees they would just see a cube roughly the size of the object it was representing. This caused even more intersection not only caused by the animation, but the layout of the proxy's themselves.

With this new pipeline, the editing of the scene is significantly faster compared to the previously employed pipeline. In the old pipeline, in the case of a necessary modification to environment, the artist would have needed to open the scene, edit it, get that approved and publish it. Then, layout and animation would have needed to be published and rendered again before the director could see that change in the rendered scene. However, with Unreal, the director could make the necessary modifications in shot and it would update across the board. Assets as well can be edited quicker, with direct feedback as to how they will look in scene.

(*a*) (*b*)

Figure 4.1: The kitchen of the animated TV series *Shane the Chef* series in Season 1 (*a*) and in Season 2 (*b*), rendered in Unreal with raytracing enabled. Raytracing allowed us to achieve a more natural lighting in the scene. (Copyright ©, Hoho Entertainment, Cloth Cat Animation)

Furthermore, real-time rendering has received a significant improvement

from the recent introduction of real-time ray-tracing. The evolution of computing power in recent years allowed video games to reach a remarkable level of graphic detail while bearing with their frame-rate requirement. Real-time ray-tracing, achieved with a dedicated chip in the latest generation of graphic cards, is yet another technique that is not confined in the non real-time domain anymore, and brings this evolution further. Figure 4.1 shows the difference between a scene with and without ray-tracing.

There are two main arguments about the importance of the ray-tracing in an animation pipeline. First of all, ray-tracing rendering allows artists to achieve more dynamic and high quality lighting effects. While non real-time systems rely on baked shadows, which look static as they don't vary as a character or camera moves in a scene, ray-tracing is able to obtain dynamic soft shadows in real time. The second advantage is the handling of reflections and refractions. In fact, with ray-tracing, artists are able to get realistic glass with real time reflections, rather than the static baked light capture that is lighting the scene.

While the advantage of this technology is evident in real-time experiences such as video games, in the production of animation this importance might be less obvious. In one case, the dynamic shadows, the advantage during the production is similar to the real-time rendering. Baking shadows is an extremely slow operation, especially in large scenes. During some stages of the production it could be avoided to bake the lights, however their presence in every stage can make immediately apparent the presence of problems with the lighting or the position of certain elements in the scene. For what concerns reflections and refraction, in addition to the time saving, the real-time ray-tracing technology makes accessible the rendering of elements that it would have been problematic to produce without. This includes scenes with glass, mirrors and these kind of elements.

In the next sections we analyse the two software that we employed for the pipeline and the motivation that led us to choose them.

4.1.1 Blender

Blender is employed for the the modelling, rigging and animation steps of the pipeline. Game engines are not currently suited for these operations, therefore an alternative was required to fill the gap. The choice of Blender let us introduce the following advantages for the pipeline.

- It supports the robustness of the pipeline: the considerable user base of Blender have a direct impact on the software. Any user can contribute in different ways, which include suggestions, bug reporting, or directly bug

fixing. Additionally, in case it would be necessary, we could directly fix problems and send the solution to Blender, which would be impossible with a non open-source software.

- It avoids lock-in: using an open-source solution avoids companies to be locked with proprietary technology or file formats.
- It is cost-effective: using open-source software instead of commercial software provides an economic advantage, as the licensing fees are expensive. The significance of the cost effectiveness of this point depends on the size of the company. To make a comparison, Autodesk Maya, one of biggest Blender competitor, is available with a subscription and it costs £1,872 per year, per machine, which for a big corporation might be neglectable, but for small or medium companies may be meaningful.

The objects created in Blender are saved both as blend and FBX files. The blend file is kept to be able to modify the object later on to create new versions, in addition to allow objects to be linked in other blend files for following steps of the pipeline. For instance, models could be linked to create rigs, and rigs could be linked to be animated. Instead, the FBX files are generated to be able to import those objects in the game engines.

Blender's support to custom scripts and addons allowed us to accommodate requests from the artists and to improve the efficiency of the system. Most of these addons are conveniences for the artists who are used to other applications, such as Autodesk Maya, to find a more familiar environment to work with. These include modifications to the user interface and addons to replicate functions from Maya. Instead, other scripts and addons have been implemented to automate operations or improve the workflow. Among others, we implemented scripts to automatically set shaders to objects or addons to help in handling layer orderings for 2D animation.

Furthermore, the open source nature of Blender has been fundamental for the implementation of the methods introduced in chapter 3 directly in the software. In fact, although either Blender and Maya support extension in Python, it is not enough to implement more complex tools. For instance, for the optimisation problems of the methods it was necessary to include the Matlab solver inside Blender. This would not have been possible with Maya.

4.1.2 Unreal Engine

For what concerns the game engine, the pipeline has been designed to be modular, thus the pipeline can be easily adapted to work with any game engine, thus using each time the engine that best suites the project. In this

initial phase of the research, we chose to work with Unreal Engine. Unreal is a renown game engine, and its adoption allowed us to gain several benefits in the design and development of the pipeline.

- Its free licence for non interactive works — such as films and tv shows — makes it cost-effective.
- It supports extensions with plugins and Python scripts, which not only simplifies the introduction of the platform in the pipeline, but also enables the integration of custom tools to provide a better experience to the artists.
- It currently has better support for ray-tracing (which is essential, for the real-time rendering) compared to its competitors. Moreover, several features of the recently presented Unreal Engine 5, including the support of "film-quality objects"[1], will be ready to be implemented as soon as this new version is released.
- It supports multi-user editing, which allows people to work at the same time at different tasks in the same scene. For instance, this function allows a user to work on the lighting while other users preview their animations. However, this feature must be well managed to avoid multiple users editing the same area, which would break the scene.

The game engine is used for the layout of the scene, the camera handling, the visual effects, weather effects, cloth simulation and all the other steps in the pipeline. Furthermore, as discussed previously, the game engine handles the rendering.

Real-time rendering is a significant advantage during production, and although offers an high graphic detail preview of the scene, there is no advantage to render a final version in real time as in Adam [Efremov, 2016]. Unreal has a built-in function to render a sequence as a video, with several settings to customize the final product. This function has been exploited to obtain the uncompressed videos without any compromise.

Unreal, in a similar way as Blender, allows the development of plugins to extend functionalities in the editor and at runtime. We exploited this possibility implementing a tool that would not have worked in a classic pipeline, to handle the localization of the shoot. This plugin can handle textures, models and also animations, for the lip sync. The users can setup an arbitrary number of languages, setup the objects that will support different localizations, and at any time the language can be switched with only one click, even when playing the scene.

[1] https://www.unrealengine.com/en-US/blog/a-first-look-at-unreal-engine-5

4.2 Development and Contribution

While game engines are an important resource for their real-time rendering capabilities, they are not adequate enough for what concerns the animation, and often they do not have any tool for modelling or rigging. Therefore, the proposed pipeline splits the tasks between two different platforms. One program, Blender, handles the modelling, rigging and animation part of the pipeline, while the game engine, Unreal, handles the lighting, layout and rendering. The communication between those two software is based on Ftrack, a tool for project management which also handles the asset versioning. Figure 4.2 illustrates the pipeline.

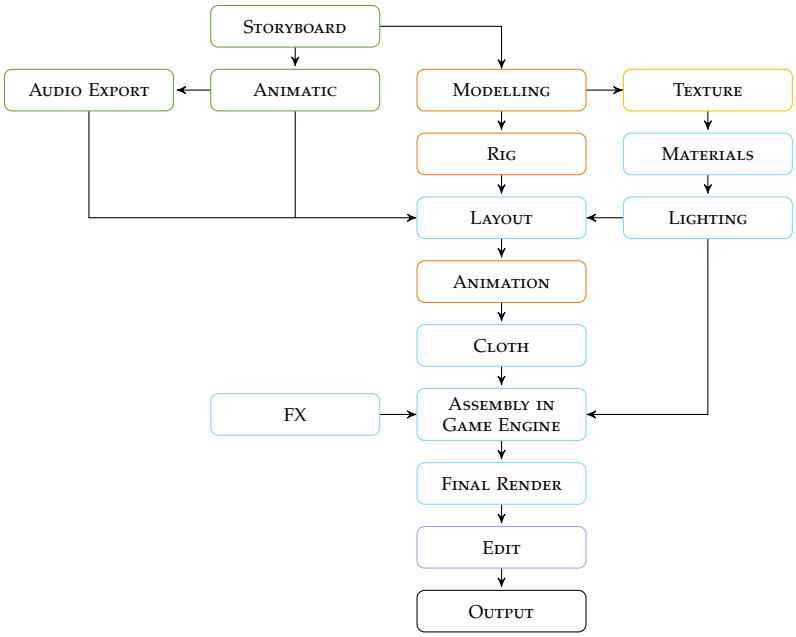


Figure 4.2: The proposed pipeline. The steps are coloured based on the following convention:
Pre-production
Performed with external tools
Editing, in Adobe Premiere
Blender
Unreal

4.2.1 Analysis of the pipeline

In this section we describe each step of the new pipeline.

Pre-production

The pre-production stage, which in the animation industry usually includes the conceptual design phase, is not different from a classic pipeline, and in-

volves the creation of the storyboard, the animatic and the audio files.

Modelling

Modelling is the first production stage of the pipeline, and takes place in Blender. The pipeline supports two different ways to create the models. From one side, it supports the classic modelling. Blender is among the most popular software for modelling, and include a comprehensive set of tools specifically for modelling. On the other hand, as shown in figure 4.4, the methods presented in section 3.2 have been integrated in Blender, allowing the artists to use existing 2D characters to generate an approximated 3D version.

Blender is a tool employed also in traditional pipelines. With the exception of the methods for the generation of 3D models from existing chartoon characters, it does not have particular differences for what concerns the modelling specifically. However, the modelling stage as a whole can be approached differently when accompanied by a game engine.

In the interviews with the artists, it has been determined that, in the previous pipeline, a large amount of models were individually modelled. This was a problem, as artists had to maintain and handle an overabundant library of assets. Elements such as plates in the kitchen of figure 4.1 are individual assets, so approximately 50 plates need to be loaded to open a file. Conversely, in Unreal you could have one plate then duplicate this in the level.

Figure 4.3: The town from the animated TV series *Shane The Chef*. (Copyright ©, Hoho Entertainment, Cloth Cat Animation)

To make another example, with the previous pipeline, each tile on every roof of a small town was modelled and therefore being seen as an individual piece of geometry. This level of detail on a roof would have been necessary only if in case of an extreme close up. There was over 52 buildings in the

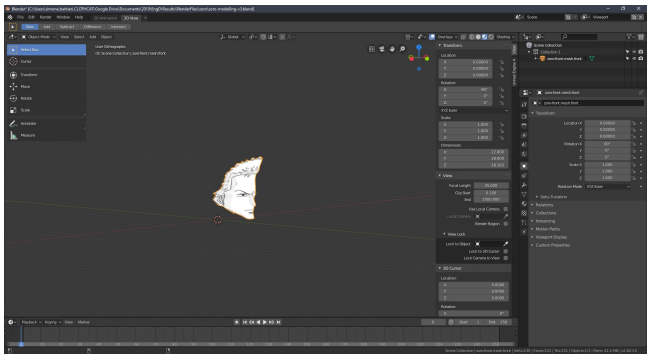


Figure 4.4: One of the models generated with method from section 3.2.2 generated within the pipeline.

town, shown in figure 4.3, each very similar, just varying in size and colour. The new pipeline makes this process more efficient, as it supports modular building, meaning the creation of a series of small assets which are combined together to create all architecture. An example can be seen in figure 4.5. This could work in this pipeline for how Unreal handles the assets. Differently from software such as Blender, or even Maya, Unreal keeps a collection of assets, and instantiate them in the scene multiple times. In Maya, the software used in the previous pipeline, all the elements are treated as different assets, and need to be imported multiple times, to be used in different part of the scene.

Models are saved as FBX — **F**ilm**b**o**x** — and blend formats for compatibility. The former is needed to be able to import the model in Unreal, as it cannot handle blend files, while the latter is kept to have a broader compatibility to work with the model during other steps of the pipeline in Blender.

Rig

The rig stage is subsequent to the modelling stage, and takes place in Blender as well. Figure 4.6 shows two characters rigged with the new system.

The system is set up in a way that the animators can control the deformation with as much detail as they need but not be overwhelmed with too many controls to animate with. Additionally, Blender's rigging system is set up in a way that is easier, compared to other software, to transfer skeletal rigs from one character to another and reposition it for characters with different proportions.

It has been determined that the best procedure to export the information required from rigs and animations is by using FBX files. However, FBX can only save information about skin weight and static shape keys — also known

Figure 4.5: Some of the modular buildings produced by assembling smaller assets. (Copyright ©, Hoho Entertainment, Cloth Cat Animation)

as morph targets or blend shapes — for geometry deformation, which limited the methods used in previous productions. Accordingly, the rigs are designed to have more deform bones than usual, and they are driven by layers of mechanical and control bones. The geometry is then weight painted in a way to comply to the limited amount of influences on vertices that can be processed in game engines. However, FBX by itself is not enough, as it supports geometry and animation, but not materials, textures and other custom information. We complement the missing information with JSON — JavaScript Object Notation — files, which are exported and imported together with the FBX files.

Textures and Materials

Textures and materials are depending on the modelling step of the pipeline, and therefore can be created in parallel with the rigging. Materials are created directly in Unreal, as it will be the platform that will take care of rendering. Textures, instead, are produced with other software, as neither Blender or Unreal offer appropriate tools to create them.

The new method employed to generate the environment, explained in the

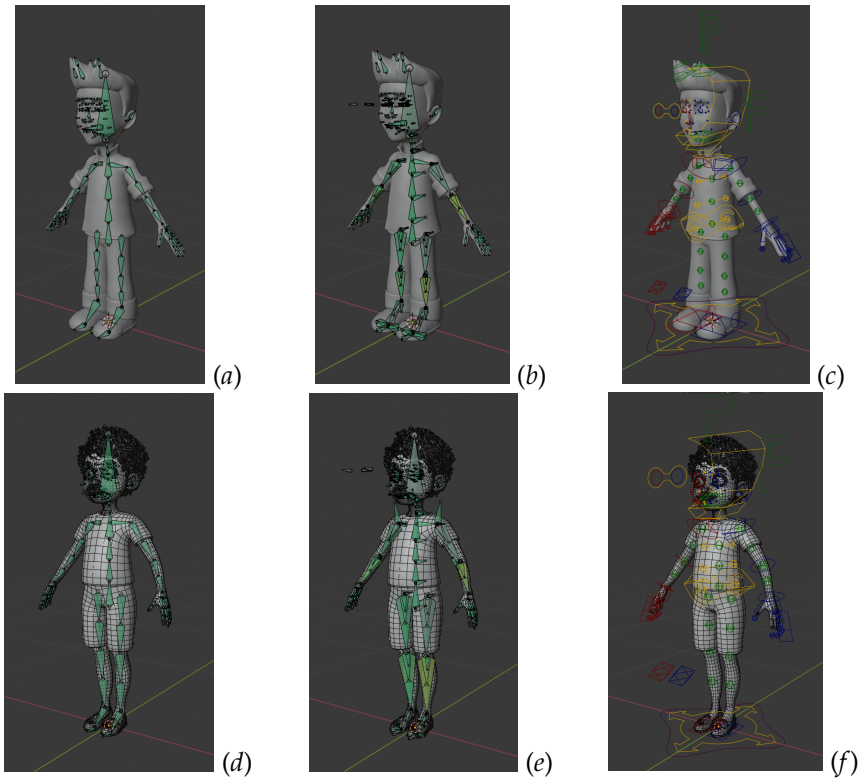


Figure 4.6: Two different characters rigged with Blender in our pipeline. First column (*a* and *d*) shows the deformation bones, the second column (*b* and *e*) show the mechanical bones and the third column (*c* and *f*) shows the control bones. (Copyright ©, Hoho Entertainment, Cloth Cat Animation)

modelling section, entails the need of a new approach for texturing dirt, grime, cracks and other similar effects onto modularly built buildings. There are ways to paint on dirt and grime in engine, although this set up is a more complex approach, in terms of configuration and resources, so it is used for more bespoke buildings only. We planned to create and use a decal pack; a collection of damage, dirt and foliage that we could add in throughout to create variation.

We also created a collection of materials ready to be applied to different models. Some of them are shown in figure 4.7. These are all instances of one material, which allows the game engine to run more efficiently. Material instances are a flexible way introduced by Unreal to have multiple materials with a concept of inheritance. An instance is generated from a single material, referred to as Parent Material. The instances inherit some properties of the parent material, while other properties are handled as parameter, that tell the instance how to function.

Figure 4.7: The collection of materials produced for the modular buildings. (Copyright ©, Hoho Entertainment, Cloth Cat Animation)

Lighting

As mentioned earlier, the support of real-time ray-tracing brings several advantages for both animation and video games, therefore is becoming popular in the industry. Our pipeline supports this new technology.

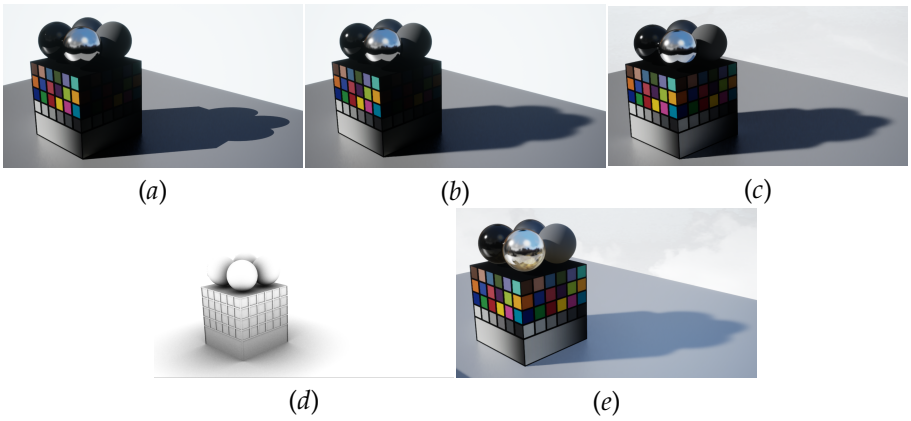


Figure 4.8: Different configurations of the lighting. (*a*): Hard shadows. (*b*): Soft shadows. (*c*): Global illumination. (*d*): Ambient occlusion. (*e*): Ambient cube maps.

An Unreal scene has a sky sphere, that represents the sky look, including details such as clouds moving and horizons.The scene also has a sky light, which applies the light of the sky to the level, matching its appearance with the light produced, including the details mentioned earlier such as the clouds, but also others closer, such as mountains. Setting the sky light as movable, it does not use any form of precomputation. We made several tests to achieve an ideal result, although the configuration will need modifications based on the projects. Figure 4.8 shows some of the tests we performed.

We have found some issues with the refraction settings in regards to refracting shadows in glass objects, with them being very strong. The only way

we found to solve this is by removing cast shadows from the object that is shadowing the glass. We also found that using a post processing volume to adjust settings like refraction and ambient occlusion on a whole scene, impacted severely the performance. Using a camera with these settings allows artists to work faster as they are only being calculated when looking through that camera.

Special Effects

The special effects available in the game engine are much wider compared to those available in the previous pipeline. The game engine makes easier adding sea, clouds, fog, fire and other effects. However in games the player is always moving and these details don't need to be so precise, therefore we needed to explore the different methods for a solution that would work for animation as well.

(*a*) (*b*)

Figure 4.9: The difference between atmospheric (*a*) and volumetric (*b*) clouds. (Copyright ©, Cloth Cat Animation)

We explored different ways of implementing fog effects in Unreal, including traditional atmospheric fog and volumetric fog and clouds. The atmospheric fog is the traditional way Unreal would treat fog, which adds a haze. The ability to use the volumetric clouds and fog imply 3D clouds that can be animated. Figure 4.9 shows this difference.

Layout

The layout stage take place in Unreal. We made this choice in consideration of the fact that while building the layout of the scene either in Blender or Unreal would have minimal impact from the artist point of view, as the interface to access assets is the same in both platforms, building the scene in Unreal gives some advantages from the pipeline point of view. First, as the final scene will be handled by Unreal, it is more appropriate to set up the layout there, as it offers to the artists a better representation of the scene. Secondly, importing

models and rigs in the scene in this stage makes easier assembling the final scene in Unreal later in the pipeline, as only animations would need to be imported from Blender, and they can be applied to the rigs straightforwardly.

Animation

Animations are produced in Blender. This stage of the pipeline requires to transfer data back from Unreal to Blender, but this is a transparent operation to the artist. In fact, the design of the pipeline makes it virtually identical to the artists in which platform they are working, at least for what concerns loading assets from the server. A detailed explanation of the system is given in section 4.2.2. The artists only need to choose the layout to load, and the system will automatically load all the assets involved in the scene, with the correct transform. Even for what concerns the data transfer itself, the data that it is necessary to transfer is minimum, as all the objects were originally created in Blender, and the reference to those object is always kept, independently of the platform. Only the information about the transform of the object in the scene is transferred.

For what concerns the actual animation, the artists can create it from scratch, using Blender's advanced toolset, or use the method introduced in section 3.3 to generate a 3D animation from an existing 2D one.

Getting the animation preview in Unreal, then, is straightforward for the artists. It requires very few inputs, to get the scene in Unreal animated and fully rendered. Animators will benefit greatly from this as they will be able to immediately see the small mistakes before sending the scene for approval or render. In the future we would like to employ the Live Link technology[2] a technique that allows the artists to link Unreal with another software and preview the animation.

For what concerns the cloth, it is handled directly in Unreal, which offers tools to handle clothing simulation in real-time.

Assembly in Game Engine

In this stage, the artists build the final scene, collecting the results of all the previous stages of the pipeline. This stage is straightforward for the artists, which have just to import all the components in the scene. Our system will automatically assign the animations to the right rig.

[2]https://docs.unrealengine.com/en-US/Engine/Animation/LiveLinkPlugin/index.html

Final Render

When production is complete, Unreal allows the user to export a sequence frame by frame. Unreal offers several options for this operation, from the frame rate, to the resolution. We export the frames as uncompressed 4K PNG files and converted to QuickTime (MOV) using Adobe Media Encoder and the PNG video codec.

As a comparison, the previous pipeline could render at lowest a frame every 20 minutes, but with an average of 30 minutes per frame. With this new pipeline, rendering with Unreal, we have been able to achieve 1 frame per second.

Edit and Output

The editing stage is performed with Adobe Premiere. The final output is then produced as Quicktime (MOV) encoded with the PNG video codec.

4.2.2 Communication between the two applications

A fundamental aspect to make the pipeline work is the communication between Blender and Unreal. To exchange information between the two applications, the formats supported by both software have been considered.

Different alternatives to share the data have been studied. In the latest years, two open-source formats in particular have become increasingly popular: Pixar's USD, Universal Scene Descriptor [Pixar Animation Studios, 2016] and the glTF, Graphics Library Transmission Format [Khronos Group, 2015]. The main benefit of these formats is their support not only of a specific elements of the scene, but of the entire scene.

Despite the benefits that these formats have, they are not yet fully supported by neither Blender nor Unreal. Moreover, as explained in section 4.2.1, we found that for several steps of the pipeline, the best way to transfer data between the two software is a combination between FBX and JSON, therefore we kept that those formats.

The first idea of the communication between Blender and Unreal is illustrated in figure 4.10. Initially, the exchange of information between the two application have been handled using sockets, and the data was transferred in real-time. An artist working in Blender was able to directly export the models in Unreal, set the layout for the environment and the initial position for the characters and send those information back to Blender. For what concerns the animation, it requires an additional step, because it needs to be baked to be imported correctly in Unreal. Moreover, this middle step was needed to take

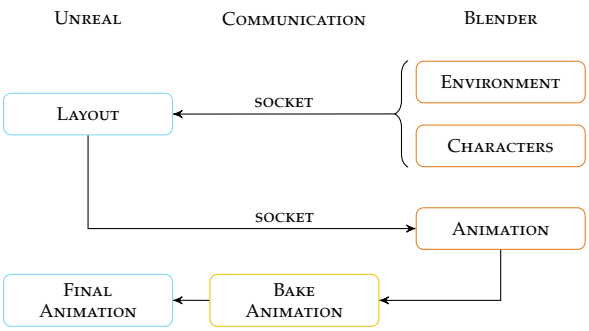


Figure 4.10: The first test of the communication system between Unreal and Blender.

in consideration the differences between Unreal and Blender, including the different unit size — 1 Unreal unit is 1 cm, while 1 Blender unit is 1 m — and the forward axis.

The implementation of this idea had been carried out using sockets implemented in Python and allowed the communication between Unreal and Blender and vice-versa in real-time. The connection between Blender and Unreal was established at the startup of either one of the applications, and the data exchanged with a combination of FBX and JSON files. However, sockets made harder to multiple artists to work simultaneously on the same projects and to handle the versioning of the models and rigs, therefore this concept has been abandoned.

A second experiment we carried out for communication consisted in exploiting the versioning system. The versioning of the objects is fundamental in a pipeline, as it allows the artists to improve models, rigs and animation over time, while keeping the previous versions available for compatibility with the existing scenes. The system has been implemented based on Ftrack, which has been used only as underlying layer. In our pipeline, data is handled in the company's own server, instead of using Ftrack's system. Ftrack has been employed as project management tool, thus its library is directly connected to the company's server to access the data.

To exploit the versioning system as a communication system, we created a common interface between Blender and Unreal to access the data, instead of letting the two application to communicate directly. Figure 4.11 illustrate this concept. Blender can save and load versions of models, rigs and animations, and load layouts. Unreal can save and load layouts and materials, and load models, rigs and animations. Blender saves data in double format, FBX and blend, the former to be able to be read by Unreal, the latter to be able to be opened and modified later in Blender. Unreal keeps the information of the

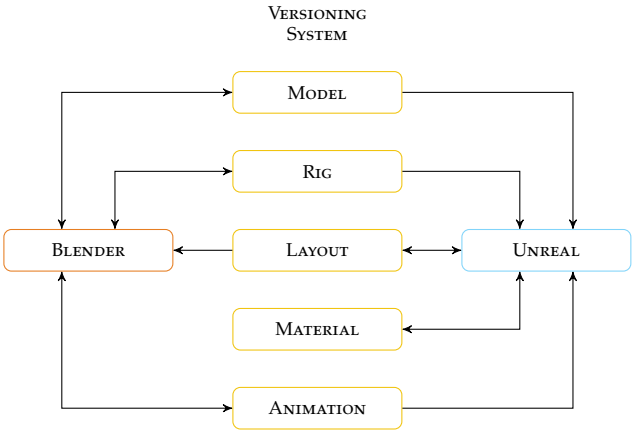


Figure 4.11: A visual explanation of how the pipeline handles the data versioning and communication.

objects that are loaded in the scene, therefore when it exports the layout, in JSON format, Blender links the original objects from the blend files.

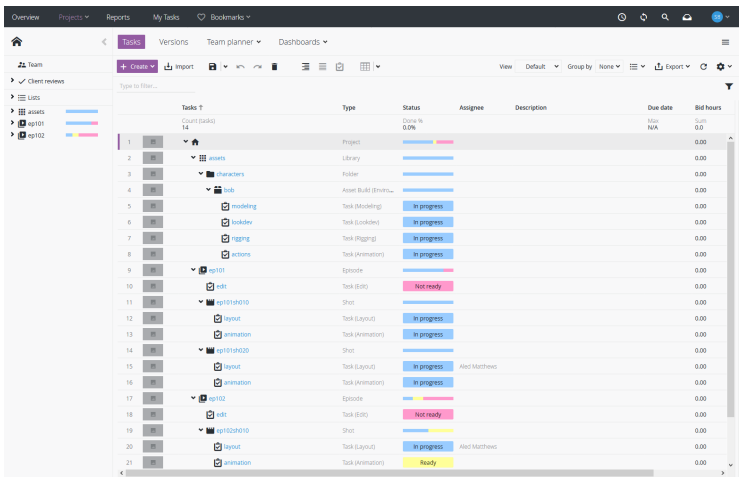


Figure 4.12: The user interface for Ftrack, where the artists can access to the tasks for the project.

The artists access the pipeline from Ftrack, shown in figure 4.12, where they can see a list of the available tasks. The director can assign certain tasks to specific artists. The artists can open the required software — Blender, Unreal or Premiere, for the editing — directly from this interface, as Ftrack support custom actions for the tasks. Tasks can be accessed directly from the software as well, through the interface in figure 4.13. The artists can also save a new version of the current work file from this window.

Once the artists have created the asset or the collection of assets that they

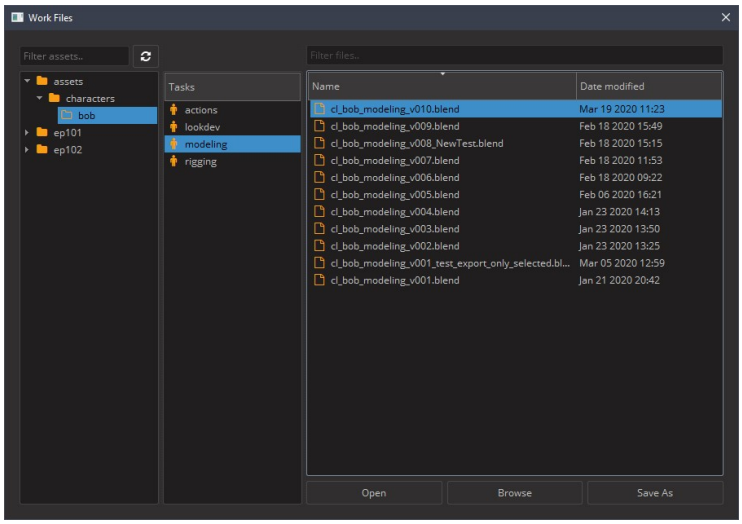
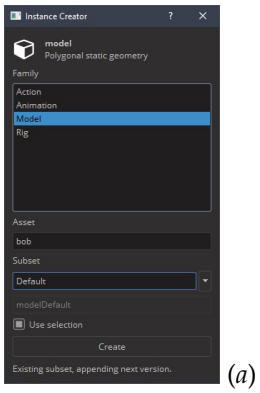
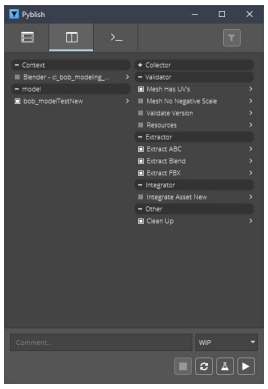


Figure 4.13: The user interface for the work files manager.



(*a*)



(*b*)

Figure 4.14: (*a*): the user interface for creating a new asset in the asset manager. (*b*): the user interface to publish the asset to the server.

want to track in the versioning system, it must be created in the asset manager, as shown in figure 4.14 (*a*). It can then be published to the server, through the interface shown in figure (*b*). The publisher supports custom validators, which help the artists to verify that the created assets have all the desired properties before they are sent for review, thus streamlining the process. The artists can also select the formats they want to use to publish the data. The export is then handled by the system, therefore the artists do not need to export the asset multiple times manually to get all the requires formats. If the asset already exists, the publisher will create a new version.

The artists can load in the scene the desired assets through the screen in figure 4.15. Loading assets this way is not loading the work file, but only importing that specific object in the current screen. The assets to load are

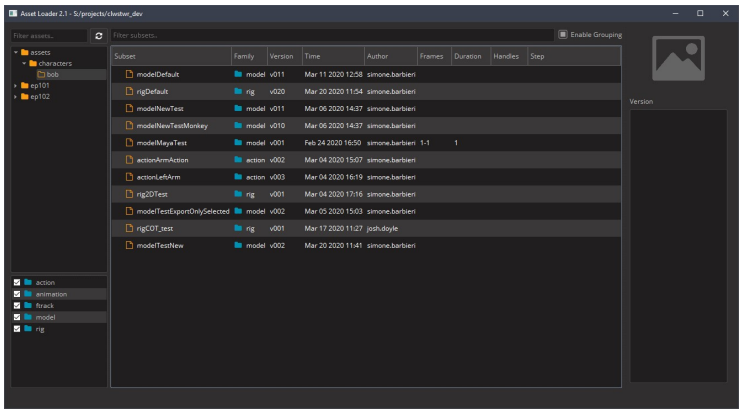
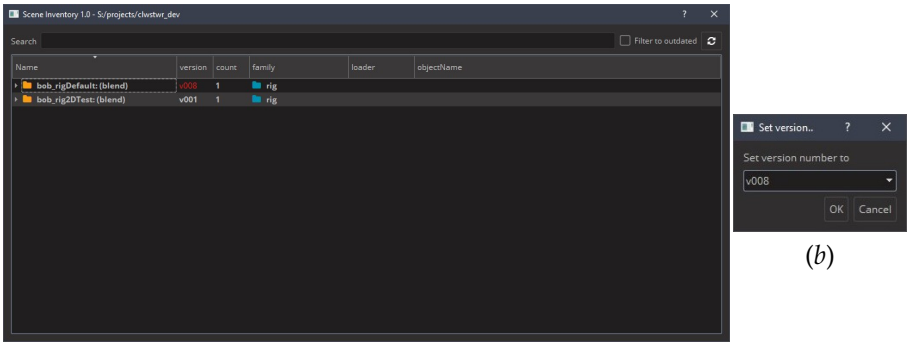


Figure 4.15: The user interface for loading assets in the scene.



(*a*) (*b*)

Figure 4.16: (*a*): the user interface for the manager of the asset currently loaded from the versioning system. (*b*): the artists can change the version of the currently loaded asset.

organized per task, and they can be filtered per type. The artists can also select which version of the asset to load. The imported assets can then be managed in the interface in figure 4.16 (*a*), where the artist can even check the asset version, which will be in red if it is not the latest version. From this screen the artist can remove the asset from the scene, or change the version, as in figure 4.16 (*b*).

Blender's linking system — also known as referencing — is not yet fully developed and cannot be employed yet for production purposes. Instead, we developed a tool that access the Ftrack library to import models, rigs or animations into a shot file. If the object was animated but needed to be updated to a newer version, the tool automatically imports the newer version, assign the animation to it and remove the old one. This, however, does mean that file sizes are larger as the objects are imported locally instead of being referenced into the file. Blender's linking system is actively being developed, so in the

future, as it become in a state usable for production, we will be able to solve the file size issue.

This second experiment has been a success, and it is the solution we adopted in the final version of the pipeline. Furthermore, exploiting the versioning system to communicate between Blender and Unreal is consistent with the flexibility we mentioned earlier, as this allow us to easily integrate in the pipeline any game engine by only allowing it to access the Ftrack library. Another advantage of this system is the ability to save straightforwardly any kind of new data, and it is automatically versioned by the system itself. This allowed us to create an action library, a repository of small animations that can be quickly loaded and applied to rigs in the scene, and combined with other animations.

4.3 Evaluation of the pipeline

This section introduces the methodology employed to test and evaluate the pipeline.

As a preliminary test, to introduce the artists to the new pipeline and to assess the usability of the system, a user test has been carried out. The test was organised to provide qualitative data on different steps of the pipeline. The users have undergone a brief training to familiarise with the new tools and the instruments to handle the assets. Then, they have been asked to complete the following six tasks:

- **Task 1**: In Blender, create a model and publish it.
- **Task 2**: In Blender, create a rig and publish it.
- **Task 3**: In Unreal, load the model and rig published in Task 1 and 2.
- **Task 4**: In Unreal, create a layout and publish it.
- **Task 5**: In Blender, load the layout, animate the rig and publish the animation.
- **Task 6**: In Unreal, load the animation.

After each task, the users have been asked to answer one question to evaluate the task difficulty, in a Likert format [Sauro and Dumas, 2009] with a score between 1 and 7, where 1 stands for *very easy* and 7 for *very hard*. Five artists participated in the test. Figure 4.17 shows the ratings for each task. The first aspect to notice is that, for every task, the average rate is below 2. This

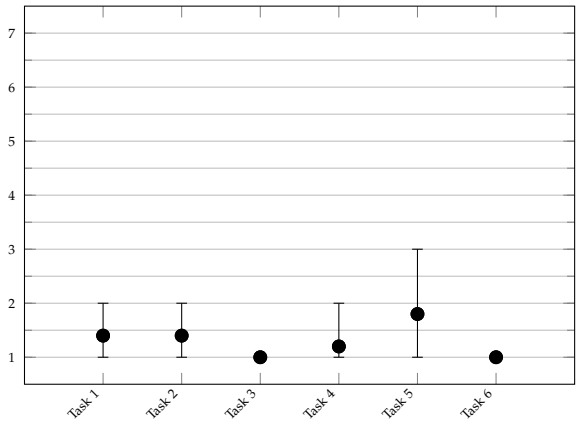


Figure 4.17: Ratings for the user evaluation of the pipeline. The minimum score is 1, while the maximum is 7. The bar represents the minimum and maximum value for each score, while the dot represents the average value.

is meaningful, as it suggests that the artists found the pipeline easy to use, which is one of the main objectives of the pipeline.

Task 5 obtained the highest score. Loading the layout in Blender causes to load all the models and rigs together, and, to prepare the publishing of the animation, the plugin creates a new collection, in the outliner, with a reference to the original object. Although this was explained in the guide provided, the rating obtained for the task suggests that some improvements might be necessary for this operation. Instead, the loading operations, excluding task 5 that included publishing the animation, obtained the lowest rating of 1. This is particularly significant for task 6, as importing animations in Unreal is not trivial, as the game engine offers numerous options, and in the proposed pipeline is a completely automatic operation, judged by the users as one of the two easiest operation.

While these preliminary tests verified the usability of the system, further testing was necessary to establish if the pipeline could work with a more complex environment, especially in anticipation of the employment of the pipeline for the production of an entire animated series. The artists have been asked to build a small scene using the new pipeline with the assets from a previous series of the studio, *Shane the Chef*, and later interviewed about the experiment. A comparison of the old and the new scene is shown in figure 4.18.

The scene took one week to be built. The first stage of the work involved bringing the original assets into the new pipeline, highlighting some of the advantages and limitations currently in the pipeline. From one side, the pipeline allowed to reduce the number of models in the project significantly. In the

Figure 4.18: A comparison between the scene from the old pipeline, on the left, and the one rebuilt with the new pipeline, on the right. (Copyright ©, Hoho Entertainment, Cloth Cat Animation)

old pipeline, in fact, it was necessary to duplicate assets that were employed multiple times. This is not required in the new pipeline, as Unreal allows the creation of multiple instances of a single model in the scene. Instead, the limitation observed did not affect the work on the test project directly, but future works with the pipeline. In fact, while the model and rigs imported, once published with the communication tool, are ready to be used for future projects as well, the materials are not, because the pipeline currently does not support them.

The second stage involved the building of the scene in Unreal. In this stage, it has been observed that the artists have been able to save time considerably, compared to the previous pipeline, as a result of the real-time rendering and the high fidelity preview. In fact, the artists were able to identify collisions or intersections between objects immediately, without needing to wait for a render, which was available only at the last stage of the pipeline. Moreover, not only it has been possible to identify problems earlier, but in the cases where it was necessary to modify an asset, the artists remained impressed about the process to update them. Once a new version of an asset is published, the artists can check the list of imported assets from a dedicated UI screen and update to the latest version — or any other version, to keep compatibility with other assets — with just one click.

Interrogated about the negative aspects of the new pipeline, the artists mentioned as "inconvenience" the necessity to use Blender instead of Maya. Considering that Blender is not as common as Maya in the industry for television and film production purposes, the available resources to learn and obtain some results have been troublesome at first. Although some resources were helpful, the initial rigging and animating procedures have been learned through a trial and error process. From this, we have begun to create our own database for best practice when rigging and animating in Blender. With this, we will be able to teach future crew members on how to use Blender more efficiently. We also designed a number of addons to expedite the rigging and animation process in Blender. To conclude, it must be considered that the pipeline has been built with modularity into account, therefore Maya can still be integrated into the pipeline at a later time, in case it is needed.

4.4 Conclusion

This chapter introduced an innovative pipeline that involves a game engine in the animation production. As the introduction of Game Engines in the animation pipeline is a relatively new idea, it is not available a large literature concerning their design. However, Epic Games, the developers of Unreal Engine, published a white paper on their real-time pipeline involving Unreal Engine, for the production of a short film *Fortnite*. A comparison has been carried out in section 2.5.3.

One of the aspects where our pipelines is superior is the communication between the software. Epic explain their process for exporting FBX from Maya and import them to Unreal, and the operation requires numerous inputs from the users. Our pipeline not only allows the users to export and import characters and objects with just a few inputs, but also integrates the versioning. Whenever something is published from Blender or Unreal, our system automatically saves it with a version number, and the user can update any object in the scene with just one click. This facilitate the maintenance of older scenes as well, as all the versions of the objects are kept.

The work on the pipeline has been carried out with the support of some artists from Cloth Cat Animation, which also aided with the testing of each stage. The testing is, however, still ongoing, and we plan to deploy the pipeline for production in 2021.

Chapter 5

Conclusion

The work presented in this thesis proposed different ways to repurpose 2D images in a 3D environment. Chapter 3 first proposed a 2.5D approach to solving the problem. 2D body parts from a turnaround are repurposed to create a 2.5D model of the character. The body parts are moved and deformed in 3D, with the employment of three techniques: billboarding, parallax scrolling and 2D shape interpolation. This method has the perk, compared to similar methods, to work with existing images instead of requiring the production of images with the tool.

However, an entirely 3D approach has two main advantages. It is more flexible in terms of the camera direction, and the interaction with the surrounding environment is easier to handle. Two different 3D approaches have been proposed. First, the thesis proposed a method that inflates the 2D meshes generated from the turnaround, based on the distance from the contour, and computes the registration between two adjacent perspectives. On the other, it presented an optimisation method, which produces a more independent 3D model due to the minimisation of a function that integrates information from the side view. Finally, the chapter proposed a system to pose a 3D character with 2D sketches. This method generates a 3D animation from an existing 2D one and, analogously to the modelling methods, has been introduced to repurpose 2D animations from previous works.

Although the 2.5D modelling method is not ready for production, it still is a first attempt at using existing 2D characters to generate a 2.5D model and has the potential to be improved in later works. For what concerns the 3D methods, both the 3D modelling and the 3D animation systems achieved the original objective of the thesis. In fact, these methods can generate an accurate representation of existing cartoon characters in 3D while keeping specific 2D traits in 3D as well. Furthermore, these methods are significantly faster

compared to modelling and animating the 3D characters from scratch. The obtained results and the evaluation carried out with the artists — section 3.5 — prove that the methods presented have the potential for their introduction in a production environment in the future, at least as a support tool for the artists, to generate a first version of the characters to be later refined.

Despite the encouraging results, there are some limitations in the system. While the surface registration allows the inflation method to preserve the 2D traits in 3D, its drawback is that, while the camera rotates around the character, it generates artifacts when switching the texture from one point of view to the next, especially with turnarounds of only four perspectives. More in-between images can reduce the artifacts, as they decrease the difference between the two adjacent perspectives. Instead, the optimisation method is unable to handle obstacles in front of the face, such as strands of hair. The formulated optimisation problem reduces the distance between two sets of points, which deforms the original flat mesh, but it cannot determine if an extrusion is necessary to match the side view. This could be a direction for future works. At the moment, a solution to this problem is to separate the obstacle and model the two parts separately and then merge them back.

For what concerns the 3D animation method, the main limitation is the consequence of the employment of the rig parameters as one of the variables of the optimisation problem. This causes the impossibility to represent poses that are not allowed by the rigging controls. If from one side this is intended to avoid poses that are not designed for the character, on the other hand, cartoons often use exaggerated or impossible poses, which cannot be matched with the presented method. Another limitation is that, using the skeleton to generate the 3D animation, the method only supports existing cutout animations, leaving all the traditional animations, or the hand-drawn elements of the cutout animation, unsupported.

Chapter 4 proposed a new unified pipeline for animation and video games. The pipeline involves a game engine, tools originally designed for video games development, but which are becoming popular for animation production as well. In fact, they provide real-time rendering, an important addition that reduces considerably the time of production. The pipeline is built upon two platforms: Blender and Unreal. Blender provides the tools to model, rig and animate the scene, and offers the methods introduced in chapter 3 to repurpose existing 2D cartoon characters in the new pipeline. Unreal is responsible of the simulations — for effects, cloth, etc. — and rendering.

From the preliminary tests performed out with the artists from Cloth Cat Animation, both the usability and the performance of the pipeline seem to be promising. The artists found the communication layer, that handles the com-

munication between Blender and Unreal and the versioning, easy to use. At the same time, the artists found the new pipeline significantly faster for the making of the scenes, in particular because of the real-time rendering offered by the game engine, which allows to identify problems early in production. The concerns of the artists about the introduction of a new tool, Blender, in place of Maya, have been addressed with custom addons to streamline the rigging and animation process, in addition to an exhaustive database of documentation and best practices for the different steps of the pipeline.

Considering the research questions asked in section 1.5, an answer can now be provided.

Research Question 1: *Is there a suitable method to represent an approximation of existing 2D cartoon characters in 3D?*

This thesis explored three different ways to represent existing 2D cartoon characters in 3D. One method is based on 2.5D modelling, while two on 3D modelling. The 2.5D method presents innovations compared to other state of the art techniques for 2.5D modelling, however this thesis found that the 3D solutions are more suitable for representing 2D cartoons in 3D. Two main reasons have been identified. On one hand, the flexibility, with the limitation to work with existing character, that a 3D solution allows, in case the top view is not provided, with the camera movement. On the other hand, its easier interaction with the 3D environment.

Research Question 2: *Is it possible to preserve essential and specific 2D features in 3D?*

This thesis proposed three methods that allow the preservation of specific 2D features in 3D, although each accomplish it in different ways. The 2.5D method maintains the features for the reason that it uses the actual 2D drawings in the 3D space. The mesh generated is deformed while the camera rotates around the character to smooth the transformation from one perspective to the next, but, nevertheless, the shape and the features are kept.

The 3D inflation method keeps the 2D features in 3D with the aid of the surface registration method. In fact, the registration identifies the correspondences of the vertices of two meshes generated from adjacent perspectives. In one of these, a feature will be visible completely, while on the other might be occluded or, because the model is just inflated, it does not have its real shape. Due to the registration, however, the system can interpolate between the two meshes while the camera rotates, showing then the features from the right perspective.

Finally, the optimization method is built to use information from the side views, and this include the specific 2D features. Moreover, the energy function include an explicit term to preserve the shape of these features from the side views.

Research Question 3: *Does the generation of approximated 3D models and animations from existing 2D cartoon characters have a benefit on the production?*

This thesis analysed the performance of the methods for the generation of 3D characters from existing 2D cartoons and of 3D animations from existing 2D ones. The analysis, carried out with the help of artists from Cloth Cat Animation studio, has revealed a significant difference in terms of time required to generate the character, compared to creating from scratch. A similar difference has been observed for the generation of the animations as well. The employments of these methods in the production, as aid for the artists to generate a first, basic version of the character, can provide a considerable benefit on the production.

Research Question 4: *Can a game engine be integrated into the production pipeline for animation?*

While historically the pipeline for video game and animation production have been different, in recent years the increase in computing power and the technological advancement have been possible to significantly improve the graphic quality of rendering in real-time. In fact, while the game engines are not a new technology, only recently industry has started using them for animation production. Chapter 4 introduced a new pipeline that integrates a game engine, Unreal Engine.

While game engines can be employed in the production for animation, they have limitations that must be tackled. First of all, currently no game engine have proper tools for creating assets, therefore they need to be complemented by other tools. In the pipeline presented in this thesis, Blender has been employed for the modelling, rigging and animation stages. The game engine, instead, is excellent for simulations, effects, and rendering. In chapter 4 there have been explained all the stages of the pipeline and the tasks handled by the game engine. Additionally, a system to handle the communication between Blender and the game engine has been introduced, which handles the versioning of the assets as well.

Research Question 5: *Is there a benefit in the integration of a game engine in the animation pipeline?*

The preliminary tests carried out with the artists from Cloth Cat Animation showed that the game engine in the pipeline allows to find errors in the scene earlier than a traditional pipeline. In fact, the real-time rendering, essential tool of the game engine, allows the artists to instantly see an accurate preview of the scene, while in the previous pipeline it was necessary to wait days to obtain the render. Moreover, the tools proposed in this thesis for transfering the assets between the software of the pipeline, that also handle their versioning, allow the user to quickly fix the problems with few interactions.

Additionally, the pipeline proposed in the thesis is designed to be used for multiple media, specifically animation and video games. Traditionally, transfer assets between two pipelines that use different software is complex and time consuming, because not all assets can be imported and it is necessary to adjust them or remake them from scratch. A pipeline that supports by design different media allows to use those assets for multiple purposes, which represents a significant time and economic saving for the studios.

5.1 Future work

This section lays out some possibilities for further research on the topic. One of the limitations that all the three methods proposed in the thesis suffered from is the requirement for some manual input from the user to mark feature points. Not only this is inconvenient for the user, but also the final results depends on the chosen points. As mentioned in section 3.4, one solution might be with the employment of optical flow and structure from motion methods. Shugrina et al. [2019] created a rich dataset for optical flow, occlusions, correspondences, etc. for non-photorealistic images and video, that could be used for this purpose. As mentioned in section 2.4, the methods presented in this thesis avoid the use of deep learning, as it is hard to train a dataset for the variety of creature that could be represented as cartoon. Nevertheless, this dataset could be tested to study the difficulty of generating the 3D models of these kind of creatures from just images.

For what concerns the new pipeline, as objective for further development, we would need to refine the rigging process further in order to be less time-consuming. One of the ways to do this would be to develop our own auto rig tool. There are auto rig tools already available for Blender, but their rig setups do not meet our specific needs and are not set up in a way that we could change to suit our needs. So creating our own tool for our specific production pipeline would be ideal and achievable.

As mentioned in section 4.1.2, the pipeline has been designed to be mod-

ular, so one of the future work include the support of another game engine, Unity. Unity and Unreal work very differently, and they have different advantages. For instance, Unity has a better support for web browsers and mobile devices, compared to Unreal, therefore, including it in the pipeline would be highly beneficial.

The modularity of the pipeline is also beneficial when considering future game engines, with special attention to Unreal Engine 5, planned for release in 2021. To integrate a new software in the pipeline only requires to incorporate it in the communication layer, which has the only requirement for the software to be able to run custom *Python* code. Epic Games has improved constantly, in the latest versions, the Unreal Engine 4 support for *Python* scripting, so it can be assumed that version 5 of the engine will support the language since the release. Epic plans to release a preview of the new version of its engine in the early 2021 [Epic Games, 2020], therefore the work to support it in the pipeline can begin in advance of the release, with the aim to offer full compatibility for the final release.

Other smaller future works include the support for new file formats, in particular glTF. At the time of the development of the pipeline, the support for glTF was very limited by both Blender and Unreal. This format would be beneficial for the pipeline as it potentially supports entire scenes, including models, rigs, materials and animations. Next, we would like to track more types of asset with the versioning system, such as textures and lights. Implementing the support for the lights would allow us to implement a light library similar to the animation library we already included in the system.

References

Ahlfors, L. V. [1953]. Complex analysis: an introduction to the theory of analytic functions of one complex variable. *New York, London*, *177*.

Akenine-Moller, T., and Haines, E. [2002]. *Real-time rendering* (2nd ed.). A. K. Peters, Ltd.

Akman, A., Sahillioglu, Y., and Sezgin, T. M. [2020]. Generation of 3d human models and animations using simple sketches.

Alexander, J. [2017]. *Star Wars: Rogue One's best character was rendered in real time, a cinema first.* `https://www.polygon.com/2017/3/1/14777806/gdc-epic-rogue-one-star-wars-k2so`.

Amidi, A. [2018]. *Disney Television Animation is exploring real-time rendering with 'Baymax Dreams'.* `https://www.cartoonbrew.com/disney/disney-television-animation-is-exploring-real-time-rendering-with-baymax-dreams-162965.html`.

An, F., Cai, X., and Sowmya, A. [2011]. Automatic 2.5D cartoon modelling. In *Image and vision computing New Zealand* (Vol. 20, pp. 149–154).

Antoine, F., Brucks, R., Karis, B., and Moran, G. [2015]. The boy, the kite and the 100 square mile real-time digital backlot. In *ACM SIGGRAPH 2015 talks on - SIGGRAPH '15.* ACM Press.

Bąk, A., and Wojciechowska, M. [2019a]. Using the game engine in the animation production process. In *Intelligent information and database systems: Recent developments* (pp. 209–220). Springer International Publishing.

Bąk, A., and Wojciechowska, M. [2019b]. Using the game engine in the an-

imation production process. In *Asian conference on intelligent information and database systems* (pp. 209–220).

Balkan, A., Dura, J., Eden, A., Monnone, B., Palmer, J. D., Tarbell, J., and Yard, T. [2003]. Parallax scrolling. In *Flash 3D cheats most wanted* (pp. 121–164). Apress.

Barbieri, S., Cawthorne, B., Xiao, Z., and Yang, X. [2017]. Repurpose 2D character animations for a VR environment using BDH shape interpolation. In *Next generation computer animation techniques* (pp. 69–85). Springer International Publishing.

Barbieri, S., Garau, N., Hu, W., Xiao, Z., and Yang, X. [2016]. Enhancing character posing by a sketch-based interaction. In *ACM SIGGRAPH 2016 posters on - SIGGRAPH '16.* ACM Press.

Barbieri, S., Jiang, T., Cawthorne, B., Xiao, Z., and Yang, X. [2018]. 3D content creation exploiting 2D character animation. In *ACM SIGGRAPH 2018 posters on - SIGGRAPH '18.* ACM Press.

Blow, J. [2008]. *Braid.* `http://braid-game.com/`.

Bradski, G. [2000]. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

Burtnyk, N., and Wein, M. [1971]. Computer-generated key-frame animation. *Journal of the SMPTE*, *80*(3), 149–153.

Catmull, E. [1978]. The problems of computer-assisted animation. In *Proceedings of the 5th annual conference on computer graphics and interactive techniques - SIGGRAPH '78.* ACM Press.

Chen, R., and Weber, O. [2015]. Bounded distortion harmonic mappings in the plane. *ACM Transactions on Graphics (TOG)*, *34*, 1 - 12.

Chen, X., Kang, S. B., Xu, Y.-Q., Dorsey, J., and Shum, H.-Y. [2008]. Sketching reality. *ACM Transactions on Graphics*, *27*(2), 1–15.

Chew, L. P. [1989]. Constrained delaunay triangulations. *Algorithmica*, *4*(1-4), 97–108.

Chien, E., Chen, R., and Weber, O. [2016]. Bounded distortion harmonic shape interpolation. *ACM Transactions on Graphics*, *35*(4), 1–15.

Choi, B., i Ribera, R. B., Lewis, J. P., Seol, Y., Hong, S., Eom, H., ... Noh, J. [2016]. SketchiMo: sketch-based motion editing for articulated characters. *ACM Transactions on Graphics*, *35*(4), 1–12.

Choi, M. G., Yang, K., Igarashi, T., Mitani, J., and Lee, J. [2012]. Retrieval and visualization of human motion data via stick figures. *Computer Graphics Forum*, *31*(7), 2057–2065.

Chui, H., and Rangarajan, A. [2003]. A new point matching algorithm for non-rigid registration. *Computer Vision and Image Understanding*, *89*(2-3), 114–141.

Cook, M. T., and Agah, A. [2009]. A survey of sketch-based 3D modeling techniques. *Interacting with Computers*, *21*(3), 201–211.

Dalstein, B., Ronfard, R., and Van De Panne, M. [2014]. Vector graphics complexes. *ACM Transactions on Graphics (TOG)*, *33*(4), 133.

Davis, J., Agrawala, M., Chuang, E., Popović, Z., and Salesin, D. [2003]. A sketching interface for articulated figure animation. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on computer animation* (pp. 320–328).

Delanoy, J., Aubry, M., Isola, P., Efros, A. A., and Bousseau, A. [2018]. 3d sketching using multi-view deep volumetric prediction. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, *1*(1), 1–22.

DeMaria, R., and Wilson, J. L. [2003]. *High score!: the illustrated history of electronic games*. McGraw-Hill Osborne Media.

Di Fiore, F., Schaeken, P., Elens, K., and Van Reeth, F. [2001]. Automatic in-betweening in computer assisted animation by exploiting 2.5D modelling techniques. In *Computer animation, 2001. the fourteenth conference on computer animation. proceedings* (pp. 192–200).

Ding, C., and Liu, L. [2016]. A survey of sketch based modeling systems. *Frontiers of Computer Science*, *10*(6), 985–999.

Duren, P. [2004]. *Harmonic mappings in the plane* (Vol. 156). Cambridge university press.

Efremov, V. [2016]. ADAM. In *ACM SIGGRAPH 2016 real-time live! on - SIGGRAPH '16.* ACM Press.

Efremov, V. [2019]. *The Heretic.* Unity Technologies.

Eitz, M., Richter, R., Boubekeur, T., Hildebrand, K., and Alexa, M. [2012]. Sketch-based shape retrieval. *ACM Transactions on Graphics*, *31*(4), 1–10.

Entem, E., Barthe, L., Cani, M.-P., Cordier, F., and Van De Panne, M. [2015]. Modeling 3D animals from a side-view sketch. *Computers & Graphics*, *46*, 221–230.

Entem, E., Parakkat, A. D., Barthe, L., Muthuganapathy, R., and Cani, M.-P. [2019]. Automatic structuring of organic shapes from a single drawing. *Computers & Graphics*, *81*, 125–139.

Epic Games. [2017]. Why real-time technology is the future of film and television production. *Epic Games whitepapers*.

Epic Games. [2020]. *A first look at Unreal Engine 5.* `https://www.unrealengine.com/en-US/blog/a-first-look-at-unreal-engine-5`.

Evans, L. C. [1998]. Partial differential equations, grad. *Studies in Math*, *19*.

Farris, J. [2020]. *Forging new paths for filmmakers on "The Mandalorian".* `https://www.unrealengine.com/en-US/blog/forging-new-paths-for-filmmakers-on-the-mandalorian`.

Felzenszwalb, P. F., and Huttenlocher, D. P. [2012]. Distance transforms of sampled functions. *Theory of computing*, *8*(1), 415–428.

Fox, T. [2015]. *Undertale.* `http://undertale.com/`.

Furusawa, C., Fukusato, T., Okada, N., Hirai, T., and Morishima, S. [2014]. Quasi 3D rotation for hand-drawn characters. In *ACM SIGGRAPH 2014 posters on - SIGGRAPH '14.* ACM Press.

Gingold, Y., Igarashi, T., and Zorin, D. [2009]. Structured annotations for 2D-to-3D modeling. *ACM Transactions on Graphics*, *28*(5), 1.

Groening, M., Brooks, J. L., and Simon, S. [1989]. *The Simpsons.* 20th Century Fox Television.

Guay, M., Cani, M.-P., and Ronfard, R. [2013]. The line of action. *ACM Transactions on Graphics*, *32*(6), 1–8.

Guay, M., Ronfard, R., Gleicher, M., and Cani, M.-P. [2015]. Space-time sketching of character animation. *ACM Transactions on Graphics*, *34*(4), 1-10.

Gunning, R. C., and Rossi, H. [2009]. *Analytic functions of several complex variables* (Vol. 368). American Mathematical Soc.

Hahn, F., Mutzel, F., Coros, S., Thomaszewski, B., Nitti, M., Gross, M., and Sumner, R. W. [2015]. Sketch abstractions for character posing. In *Proceedings of the 14th ACM SIGGRAPH / eurographics symposium on computer animation - SCA '15.* ACM Press.

Han, X., Gao, C., and Yu, Y. [2017]. Deepsketch2face: a deep learning based sketching system for 3d face and caricature modeling. *ACM Transactions on graphics (TOG)*, *36*(4), 1–12.

Harary, F. [1969]. *Graph theory*. Avalon Publishing.

Haswell, H. [2014]. To infinity and back again: hand-drawn aesthetic and affection for the past in Pixar's pioneering animation. *Alphaville: Journal of Film and Screen Media*, *8*(2).

Igarashi, T., and Hughes, J. F. [2001]. A suggestive interface for 3D drawing. In *Proceedings of the 14th annual ACM symposium on user interface software and technology - UIST '01.* ACM Press.

Igarashi, T., Matsuoka, S., and Tanaka, H. [1999]. Teddy. In *Proceedings of the 26th annual conference on computer graphics and interactive techniques - SIGGRAPH '99.* ACM Press.

Igarashi, T., and Mitani, J. [2010]. Apparent layer operations for the ma-

nipulation of deformable objects. *ACM Transactions on Graphics*, *29*(4), 1.

Jacobson, A., Baran, I., Popovic, J., and Sorkine, O. [2011]. Bounded biharmonic weights for real-time deformation. *ACM Trans. Graph.*, *30*(4), 78.

Jacobson, A., Weinkauf, T., and Sorkine, O. [2012]. Smooth shape-aware functions with controlled extrema. In *Computer graphics forum* (Vol. 31, pp. 1577–1586).

Jiang, T., Qian, K., Liu, S., Wang, J., Yang, X., and Zhang, J. [2017]. Consistent as-similar-as-possible non-isometric surface registration. *The Visual Computer*, 1–11.

Jost, J., Jost, J., and Li-Jost, X. [1998]. *Calculus of variations* (Vol. 64). Cambridge University Press.

Kajisa, K. [2016]. *The Gift.* Marza Animation Planet.

Kajisa, K. [2017]. *Ultimate Bowl 2017.* Marza Animation Planet.

Karpenko, O. A., and Hughes, J. F. [2006]. SmoothSketch. *ACM Transactions on Graphics*, *25*(3), 589.

Kay, R. [2018]. *The future of 2D gaming.* `https://www.gamesindustry.biz/articles/2018-11-27-the-future-of-2d-gaming`.

Khan, U. Z. [2018]. *Allahyar and the Legend of Markhor.* 3rd World Studios.

Khronos Group. [2015]. *GL Transmission Format Website.* `http://www.khronos.org/gltf/`.

King, D. [2015]. *Unreal encourages filmmakers to use its game engine.* `https://www.cartoonbrew.com/tech/unreal-encourages-filmmakers-to-use-its-game-engine-111708.html`.

Kitamura, M., Kanamori, Y., and Tsuruno, R. [2015]. 2.5D modeling from illustrations of different views. *International Journal of Asia Digital Art and Design*, *8*(4), 74–79.

Konno, H., and Miyamoto, S. [1996]. *Mario Kart 64.* Nintendo.

Kraevoy, V., Sheffer, A., and van de Panne, M. [2009]. Modeling from contour drawings. In *Proceedings of the 6th eurographics symposium on sketch-based interfaces and modeling - SBIM '09.* ACM Press.

Kruger, G. [2018]. *Why indie games are important for the gaming industry.* `https://memeburn.com/gearburn/2018/07/indie-games-industry-importance/`.

Lasseter, J., Arnold, B., and Guggenheim, R. [1995]. *Toy Story.* Buena Vista Pictures.

Lear, J., Scarle, S., and McClatchey, R. [2019]. Asset pipeline patterns: patterns in interactive real-time visualization workflow. In *Proceedings of the 24th european conference on pattern languages of programs* (pp. 1–11).

Lee, S., Feng, D., Grimm, C., and Gooch, B. [2008]. A sketch-based user interface for reconstructing architectural drawings. *Computer Graphics Forum*, *27*(1), 81–90.

Li, C., Pan, H., Liu, Y., Tong, X., Sheffer, A., and Wang, W. [2018]. Robust flow-guided neural prediction for sketch-based freeform surface modeling. *ACM Transactions on Graphics (TOG)*, *37*(6), 1–12.

Lin, J., Igarashi, T., Mitani, J., Liao, M., and He, Y. [2012]. A sketching interface for sitting pose design in the virtual environment. *IEEE Transactions on Visualization and Computer Graphics*, *18*(11), 1979–1991.

Lipkin, N. [2012]. Examining indie's independence: The meaning of "indie" games, the politics of production, and mainstream cooptation. *Loading...*, *7*(11).

Liu, R., Zhang, H., Shamir, A., and Cohen-Or, D. [2009]. A part-aware surface metric for shape analysis. *Computer Graphics Forum*, *28*(2), 397–406.

Liu, X., Mao, X., Yang, X., Zhang, L., and Wong, T.-T. [2013]. Stereoscopizing cel animations. *ACM Transactions on Graphics*, *32*(6), 1–10.

Mao, C., Qin, S. F., and Wright, D. K. [2005]. A sketch-based gesture interface for rough 3D stick figure animation. *Eurographics Workshop on Sketch-Based Interfaces and Modeling*.

Marr, D. [1982]. *A computational investigation into the human representation and processing of visual information*. The MIT Press.

McCann, J., and Pollard, N. [2009]. Local layering. In *ACM SIGGRAPH 2009 papers on - SIGGRAPH '09*. ACM Press.

Mikolajczyk, K., Leibe, B., and Schiele, B. [2005]. Local features for object class recognition. In *Tenth IEEE international conference on computer vision (ICCV'05) volume 1*. IEEE.

Miyazaki, H. [2001]. *Spirited Away*. Studio Ghibli.

Moldenhauer, C., and Moldenhauer, J. [2017]. *Cuphead*. `http://cupheadgame.com/`.

Morris, A. [2018]. *How the rise of indie games has revitalized the video game industry*. `https://www.allbusiness.com/indie-games-video-game-industry-101485-1.html`.

Muller, M. [2017]. *Adam – step by step*. `https://blogs.unity3d.com/2017/01/11/adam-step-by-step/`.

Nealen, A., Igarashi, T., Sorkine, O., and Alexa, M. [2007]. FiberMesh. *ACM Transactions on Graphics*, *26*(3), 41.

Nealen, A., Sorkine, O., Alexa, M., and Cohen-Or, D. [2005]. A sketch-based interface for detail-preserving mesh editing. *ACM Transactions on Graphics*, *24*(3), 1142.

Olsen, L., Samavati, F., and Jorge, J. [2011]. Naturasketch: Modeling from images and natural sketches. *IEEE Computer Graphics and Applications*, *31*(6), 24–34.

Olsen, L., Samavati, F. F., Sousa, M. C., and Jorge, J. A. [2009]. Sketch-based modeling: A survey. *Computers & Graphics*, *33*(1), 85–103.

Pereira, J. P., Jorge, J. A., Branco, V., and Ferreira, F. N. [2000]. Towards calligraphic interfaces: sketching 3D scenes with gestures and context icons. *Václav Skala - UNION Agency*.

Pinkall, U., and Polthier, K. [1993]. Computing discrete minimal surfaces and their conjugates. *Experimental mathematics*, *2*(1), 15–36.

Pixar Animation Studios. [2016]. *Universal Scene Description Website.* `http://www.openusd.org`.

Pohl, B. J., Brakensiek, T., and Lombardo, S. [2018]. Fortnite trailer: Developing a real-time pipeline for a faster workflow. *Epic Games whitepapers*.

Poranne, R., and Lipman, Y. [2014]. Provably good planar mappings. *ACM Transactions on Graphics*, *33*(4), 1–11.

Prasad, L. [1997]. Morphological analysis of shapes. *CNLS newsletter*, *139*(1), 1997–07.

Ramos, L. A. G. T. [2017]. *Production of 3D animated short films in Unity 5: Can game engines replace the traditional methods?* Universidade Católica Portuguesa.

Rivers, A., Igarashi, T., and Durand, F. [2010]. 2.5D cartoon models. *ACM Transactions on Graphics*, *29*(4), 1.

Sauro, J., and Dumas, J. S. [2009]. Comparison of three one-question, post-task usability questionnaires. In *Proceedings of the sigchi conference on human factors in computing systems* (pp. 1599–1608).

Savov, V. [2017]. *Unity is the little game engine that could revolutionize animated movies.* `https://www.theverge.com/2017/6/30/15899446/unity-cinemachine-unite-europe-2017-animation`.

Scherba, T. [2016]. *Virtual reality is about to go mainstream, but a lack of content threatens to hold it back.* `https://techcrunch.com/2016/04/03/virtual-reality-is-about-to-go-mainstream-but-a-lack-of-content-threatens-to-hold-it-back/`.

Schick, C. [2020]. *Love & 50 Megatons.* Film Academy Baden-Württemberg.

Shewchuk, J. R. [1996, may]. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In M. C. Lin and D. Manocha (Eds.), *Applied computational geometry: Towards geometric engineering* (Vol. 1148, pp. 203–222). Springer-Verlag. (From the First ACM Workshop on Applied Computational Geometry)

Shtof, A., Agathos, A., Gingold, Y., Shamir, A., and Cohen-Or, D. [2013]. Geosemantic snapping for sketch-based modeling. *Computer Graphics Forum*, *32*(2pt2), 245–253.

Shugrina, M., Liang, Z., Kar, A., Li, J., Singh, A., Singh, K., and Fidler, S. [2019]. Creative flow+ dataset. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 5384–5393).

Sivic, J., Russell, B., Efros, A., Zisserman, A., and Freeman, W. [2005]. Discovering objects and their location in images. In *Tenth IEEE international conference on computer vision (ICCV'05) volume 1.* IEEE.

Sorkine, O., and Cohen-Or, D. [2004]. Least-squares meshes. In *Shape modeling applications, 2004. proceedings* (pp. 191–199).

Sporn, M. [2009]. *How to draw Mickey.* `http://www.michaelspornanimation.com/splog/?p=1939`.

Sýkora, D., Sedlacek, D., Jinchao, S., Dingliana, J., and Collins, S. [2010]. Adding depth to cartoons using sparse depth (in)equalities. *Computer Graphics Forum*, *29*(2), 615–623.

Takahashi, D. [2013]. *How Pixar made Monsters University, its latest technological marvel.* `https://venturebeat.com/2013/04/24/the-making-of-pixars-latest-technological-marvel-monsters-university/`.

Takahashi, D. [2017]. *District 9 director Neill Blomkamp hopes game engines can democratize film.* `https://venturebeat.com/2017/10/08/district-9-director-neill-blomkamp-hopes-game-engines-can-democratize-film/`.

Toftedahl, M., and Engström, H. [2019]. A taxonomy of game engines and the tools that drive the industry. In *Digra 2019, the 12th digital games research association conference, kyoto, japan, august, 6-10, 2019.*

Trousdale, G., Wise, K., and Hahn, D. [1991]. *Beauty and the Beast.* Buena Vista Pictures.

Wei, X. K., and Chai, J. [2011]. Intuitive interactive human-character posing with millions of example poses. *IEEE Computer Graphics and Applications*, *31*(4), 78–88.

Wells, S. [2018]. *Disney to debut its first VR short next month.* `https://techcrunch.com/2018/07/19/disney-to-debut-its-first-vr-short-next-month/`.

Wiley, K., and Williams, L. R. [2006]. Representation of interwoven surfaces in 2 1/2 D drawing. In *Proceedings of the SIGCHI conference on human factors in computing systems - CHI '06.* ACM Press.

Williams, L. R., and Hanson, A. R. [1996]. Perceptual completion of occluded surfaces. *Computer Vision and Image Understanding*, *64*(1), 1–20.

Xu, B., Chang, W., Sheffer, A., Bousseau, A., McCrae, J., and Singh, K. [2014]. True2Form. *ACM Transactions on Graphics*, *33*(4), 1–13.

Xu, K., Chen, K., Fu, H., Sun, W.-L., and Hu, S.-M. [2013]. Sketch2Scene. *ACM Transactions on Graphics*, *32*(4), 1.

Yeh, C.-K., Jayaraman, P. K., Liu, X., Fu, C.-W., and Lee, T.-Y. [2015]. 2.5D cartoon hair modeling and manipulation. *IEEE Transactions on Visualization and Computer Graphics*, *21*(3), 304–314.

Yeh, I.-C., Lin, C.-H., Sorkine, O., and Lee, T.-Y. [2011]. Template-based 3d model fitting using dual-domain relaxation. *IEEE Transactions on Visualization and Computer Graphics*, *17*(8), 1178–1190.

You, L., Yang, X., Pan, J., Lee, T.-Y., Bian, S., Qian, K., ... Zhang, J. J. [2020]. Fast character modeling with sketch-based pde surfaces. *Multimedia Tools and Applications*.

Zanni, C., Bernhardt, A., Quiblier, M., and Cani, M.-P. [2013]. SCALe-invariant integral surfaces. *Computer Graphics Forum*, *32*(8), 219–232.

Zeleznik, R. C., Herndon, K. P., and Hughes, J. F. [1996]. SKETCH. In

Proceedings of the 23rd annual conference on computer graphics and interactive techniques - SIGGRAPH '96. ACM Press.

Zemeckis, R., Marshall, F., and Watts, R. [1988]. *Who Framed Roger Rabbit.* Buena Vista Pictures.

Zhang, L., Huang, H., and Fu, H. [2012]. EXCOL: An EXtract-and-COmplete layering approach to cartoon animation reusing. *IEEE Transactions on Visualization and Computer Graphics*, *18*(7), 1156–1169.

Zimmermann, J., Nealen, A., and Alexa, M. [2007]. SilSketch. In *Proceedings of the 4th eurographics workshop on sketch-based interfaces and modeling - SBIM '07.* ACM Press.