

Semantic-Aware Animation of Hand-Drawn Characters

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Visual Computing

eingereicht von


Thorsten Korpitsch, BSc

Matrikelnummer 01529243

an der Fakultät für Informatik
der Technischen Universität Wien

Betreuung: Eduard Gröller, Univ.Prof. Dipl.-Ing. Dr.techn.
Mitwirkung: Hsiang-Yun Wu, PhD

Wien, 6. August 2023


Thorsten Korpitsch


Eduard Gröller

Semantic-Aware Animation of Hand-Drawn Characters

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Visual Computing

by

Thorsten Korpitsch, BSc

Registration Number 01529243

to the Faculty of Informatics

at the TU Wien

Advisor: Eduard Gröller, Univ.Prof. Dipl.-Ing. Dr.techn.

Assistance: Hsiang-Yun Wu, PhD

Vienna, 6th August, 2023


Thorsten Korpitsch


Eduard Gröller

Erklärung zur Verfassung der Arbeit

Thorsten Korpitsch, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 6. August 2023



Thorsten Korpitsch



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Danksagung

Ich möchte meinen Betreuern Meister Eduard Gröller, Univ.Prof. Dipl.-Ing. Dr.techn., und Hsiang-Yun Wu, Ph.D., für ihre Geduld und ihre unermüdliche Unterstützung mit Rat, Tat und Anleitung während der Durchführung und Erstellung dieser Arbeit danken. Ohne ihre unbezahlbaren Ratschläge wäre diese Arbeit nicht möglich gewesen.

Ich möchte Valentina, meiner besseren Hälfte, dafür danken, dass sie mich während meiner gesamten Studienzzeit immer unterstützt hat, und für ihre unendliche Geduld und Freundlichkeit, die das Leben viel lebenswerter macht. Außerdem möchte ich meinen Eltern und meiner Großmutter für ihre unermüdliche Unterstützung danken, nicht nur während des Schreibens dieser Arbeit, sondern während meiner gesamten Studienzzeit, davor und danach.

Zu guter Letzt möchte ich mich bei meinen Freunden und anderen Verwandten für ihre Unterstützung bedanken, insbesondere bei Simon und David, meinen Studienkollegen, mit denen wir mit viel Spaß durch die Masterkurse gestürmt sind und denen jede Herausforderung, die wir während des Studiums zu meistern hatten, viel leichter fiel, als wenn wir sie alleine bewältigt hätten.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Acknowledgements

I want to thank my advisors Meister Meister Eduard Gröller, Univ.Prof. Dipl.-Ing. Dr.techn., and Hsiang-Yun Wu, Ph.D., for their patience and tirelessly providing me with advice, action, and guidance during the implementation and writing of this thesis. Without the priceless advice, the thesis would not have been possible.

I want to thank Valentina, my better half, for always supporting me throughout my years in university and for her never-ending patience and kindness making life a lot more liveable. Further, I want to thank my parents and grandmother for their tireless support not only during the writing of this thesis but throughout my whole years in university, before that and after graduation.

Last but not least I want to thank my friends and other relatives for their support, especially Simon and David my study colleagues with whom we blazed through the master's courses with a lot of fun and all challenges we faced during studying seemed just a lot more possible than when facing them alone.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Kurzfassung

In den letzten Jahren wurde viel auf dem Gebiet der Bildungsunterhaltung geforscht, das effektive Lernprozesse erleichtert, indem es das Engagement der Lernenden erhöht. Geführte Visualisierungen, wie etwa audiogeführte Museumsführungen oder AR-geführte Stadtführungen, sind eine der möglichen Anwendungen. Geführte Visualisierungen sind eine Form der mentalen Übung, die traditionell eine verbale Anleitung beinhaltet, die den Benutzer durch eine Reihe von Visualisierungen führt. Mittels neuer Technik wie Augmented Reality können zusätzliche Informationen integriert werden, wie beispielsweise eine virtuelle Figur, welche die BenutzerInnen führt, was eine fesselnde Benutzererfahrung ermöglicht. In dieser Arbeit wollen wir einen ersten Schritt in Richtung geführter Visualisierung machen, indem wir eine handgezeichnete Figur zu Anleitungszwecken einführen. Wir konzentrieren uns besonders auf die Animation, da Charakteranimationen in verschiedenen Anwendungen, wie z.B. der Computergrafik, verwendet werden, aber ohne gewisse Vorkenntnisse für BenutzerInnen schwer selbst zu generieren sind. Hier stellen wir eine neuartige Prozesskette zur automatischen Generierung passender Animationen für handgezeichnete Charaktere vor. Der Ansatz besteht aus fünf Schritten. (1) Die handgezeichnete Figur wird aus einem Eingabebild erkannt, und (2) die Teile der gezeichneten Figur, wie z. B. die Beine und der Kopf, werden jeweils identifiziert. (3) Ein Knochenskelett für die Animation wird extrahiert und mit den semantischen Informationen aus dem vorherigen Schritt ergänzt. (4) Auf der Grundlage des erweiterten Skeletts ordnen wir eine Superklasse zu, zu der das Skelett gehört, z.B. Vierbeiner, Fliegende, oder Humanoide, und überlagern die Endeffektoren, wie beispielsweise die Beine, des Skeletts mit den Endeffektoren des Referenzskeletts der Superklasse. (5) Schließlich erzeugen wir ein Dreiecksnetz aus dem Eingabebild. Sobald das passende Referenzskelett und die handgezeichnete Figur überlagert sind, ist die Figur animiert und kann Benutzer in verschiedenen Anwendungen ansprechen. Um die Machbarkeit unseres Ansatzes zu zeigen, evaluieren wir die vorgeschlagene Prozesskette mit einer Reihe von handgezeichneten Figuren.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Abstract

In recent years there has been a lot of research in the area of edutainment, which facilitates effective learning processes by increasing the engagement of the learners. Guided visualisations, such as audio-guided museum tours or Augmented Reality-guided city tours, are one of the potential applications. Guided visualisations are a form of mental practice which traditionally involves verbal guidance that guides a user through a series of visualisations. With the technique of Augmented Reality, one can integrate additional information to guide users or embody verbal guidance with a virtual character, which enables an engaging experience. In this thesis, we aim to make a first step towards guided visualisation by introducing a hand-drawn character for instruction purposes. We especially focus on animation, since character animations are used in different applications, such as computer graphics, but can be hardly generated without certain pre-knowledge. Here, we present a novel pipeline for automatically generating believable movements for hand-drawn characters. The approach consists of five steps. (1) the hand-drawn character is detected from an input image, and (2) the sub-parts of the drawn character, such as the legs and the head, are identified, respectively. (3) A bone skeleton for animation is extracted and augmented with the semantic information from the previous step. (4) Based on the augmented skeleton, we assign a super-class that the skeleton belongs to, i.e., quadruped, flying or humanoid, and match the end-effectors of the skeleton to the end-effectors of the reference skeleton of the super-class. (5) Finally, we generate a triangular mesh from the input illustration. Once the matching reference skeleton and the hand-drawn character are overlaid, the character is animated and can attract users in different applications. To show the feasibility of our approach, we evaluate the proposed pipeline with a set of hand-drawn characters showing several well-articulate drawings.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Contents

Kurzfassung	xi
Abstract	xiii
Contents	xv
1 Introduction	1
1.1 Background	1
1.2 Motivation	2
1.3 Problem Statement and Challenges	2
1.4 Goals and Contributions	3
1.5 Thesis Structure	3
2 Related Work	5
3 Technical Background and Definitions	13
3.1 Machine Learning, Neural Networks, and Deep Learning	13
3.2 Graph Theory	19
3.3 Character Animation	21
3.4 Creating Skeletons	28
4 Methodology of Animating Hand-Drawn Characters	31
4.1 Overview of the Animation Pipeline	31
4.2 Character Detection	31
4.3 Semantic Segmentation	37
4.4 Character Skeletonisation	39
4.5 Character Identification	44
4.6 Rigging and Animation	49
5 Prototype Implementation	59
6 Results and Discussion	63
6.1 Experimental Results	63
6.2 Evaluation and Discussion	68
	xv

6.3	Limitations of the Pipeline	68
7	Conclusion and Future Work	75
7.1	Summary and Conclusion	75
7.2	Future Work	76
	List of Figures	77
	List of Tables	81
	List of Algorithms	83
	Bibliography	85

Introduction

In the following chapters, first, the background and the motivation for this thesis are described. Further, the goals and challenges as well as the contribution of this thesis are explained. The structure of the remainder of this thesis is described last.

1.1 Background

Hand-drawn characters, especially children's drawings, have been of interest to researchers for a very long time and have received greater attention in recent years. There have been many works focusing on analysing child's drawings such as Rémi et al. [RFC02], Pysal et al. [PASA21], Thomas et al. [TPP⁺22], and Beltzung et al. [BPRS23] in different fields of research from computer science to psychology or medical science.

Another interesting field of research that relates to animated characters is guided visualisations, such as the CHARIOT program [Sta]. While this program contains many different approaches, such as games or interactive stories, one interesting approach is combining edutainment with animated characters. Animated characters explain the medical procedures to calm child patients in advance. This can be described as a form of edutainment.

The term edutainment is a combination of the words education and entertainment. The general idea is that any form of media, content, or activities are designed to educate and entertain at the same time. The main goal of edutainment is to give an opportunity for acquiring knowledge in an interesting and engaging way. Anikina and Oksana [AY15] describe it as a feature of modern technology implementation in traditional lectures and classes. Edutainment can be presented in many different forms such as, but not limited to, television shows, video games, mobile apps, interactive websites, and toys. In recent years edutainment received a lot of attention in research. One research area in edutainment is physicalisation such as presented by Schindler et al. [SKRW22] and Raidou

et al. [RGW20] for anatomical and biological edutainment. The goal is to facilitate the process of learning by increasing users' understanding and engagement with the topics at hand [KM15].

Another area that gained a lot of research interest in recent years is using Augmented and Virtual Reality (AR, VR) in edutainment. Philipp et al. [SWO⁺14] present an Augmented Reality system for learning bone anatomy. Similar to previous work the hypothesis is that due to a higher level of engagement of users the learning effect can be improved.

1.2 Motivation

The main idea before this thesis was to evaluate the impact of using guided AR visualisation in the area of edutainment.

Instead of using a predefined character to embody the guide we use the user's own hand-drawn character for guidance through the visualisation. The hypothesis is that the effect of constructivism described by Huang et al. [HRL10], can increase the engagement of the user and facilitate the learning process.

To achieve a high level of engagement and immersion, the animation of hand-drawn characters should be believable. This means that the different parts of the drawing should be considered when generating the motion, such that i.e., the limbs of a drawn animal are used for walking, rather than merely moving any extremities or transferring any reference motion to the hand-drawn character. We refer to this information as the semantics of the drawing.

Since no pipeline exists, to the best of our knowledge, that is able to animate hand-drawn characters semantic-aware, which creates believable motion, this thesis focuses on the creation of such a pipeline. The impact of guided AR visualisation is evaluated in future work and therefore this work can be seen as the first step towards a system for immersive guided visualisation.

1.3 Problem Statement and Challenges

The objective of this thesis is to provide a fully automated pipeline for the semantic-aware animation, in 2.5D, of arbitrary hand-drawn (2D) characters.

The following challenges need to be overcome in order to solve the stated problem:

- **Detection:** In the first step given an RGB image as an input, the hand-drawn character needs to be detected for further processing.
- **Segmentation:** In the second step, the sketch is segmented based on the semantics of the sketch, i.e., identifying which parts of the sketch are representing limbs, and which parts are representing the body or the head.

- **Skeletonisation:** Afterwards the sketch is skeletonised to create the basis for the animation, which is skeleton-based.
- **Identification:** Then the skeleton needs to be matched to the closest reference skeleton that exists, making it possible to get the best matching animation for the given input.
- **Rigging and Animation:** As a last step, the hand-drawn character is triangulated and the resulting mesh is rigged and assigned an animation.

1.4 Goals and Contributions

The goal of this thesis is to provide a fully automated pipeline that animates a hand-drawn character based on the semantic information of the drawing. The input of the pipeline is defined as an image containing a hand-drawn character. Based on the given input the pipeline then follows the steps described in Chapter 1.3 to compute a semantic aware animation of the character. Results show that on the tested set of hand-drawn characters the pipeline leads to promising outcomes. This allows future work to investigate the impact of edutainment when using guided visualisations with hand-drawn characters on the user experience, with a special focus on child education.

The contributions of the thesis can be summarised as follows:

- Pipeline architecture for fully automated semantic-aware animation of hand-drawn characters
- Proposal for identifying a believable motion type by matching semantic skeletons
- Evaluation of the proposed algorithms using a set of hand-drawn characters
- Proposal for adding label information to the Geometric Graph Distance proposed by Cheong et al. [CGK⁺09]

1.5 Thesis Structure

The remainder of the thesis is structured as follows. Chapter 2 provides an overview of previous work done related to hand-drawn characters and character animation. Chapter 3 explains key concepts and contains definitions and technical background for algorithms and concepts, such as graph theory, machine learning, character animation, and skeletonisation, used in the following chapters of the thesis. Chapter 4 gives an overview of the proposed pipeline and describes each necessary step in detail towards generating an animated hand-drawn character. Chapter 5 elaborates implementation details of the methods described in the previous chapter. Chapter 6 shows the results of the implementation and discusses its limitations. Finally Chapter 7 summarises the contributions and provides an outlook on future work.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Related Work

This chapter briefly discusses related work focusing specifically on research related to hand-drawn characters and their animation. Further, it describes differences between previous research and the proposed approach.

Lately, there has been increasing interest in the animation of hand-drawn characters that are available to the broad public rather than specialists. Yamada et al. [Jun] present in their works an augmented reality system that can detect hand-drawn characters on a sheet of paper. First, a character is detected as seen in Figure 2.1(a). The detected hand-drawn character is then transformed into a 3D model and is animated by randomly moving extremities of the 3D model as shown in Figure 2.1. Multiple characters can be detected and transformed into 3D models at the same time, which creates a lively environment. Further, Figure 2.1(c) shows users can interact with the characters, i.e., dropping objects that the characters start interacting with.

Contrary to the work of Yamada et al. [Jun] we propose to avoid animating the characters in a random way. Depending on the characteristics of the hand-drawn character, we determine a fitting type of movement. The giraffe and the panda in Figure 2.1 should move differently since the panda will walk on two legs, while the giraffe with four legs should have quadruped movement to be more immersive.

Smith et al. [SZL⁺23] present a system that is capable of automatically animating a child’s drawing simulating a human figure. Their proposed approach is robust to the variety of different depictions of humans when it comes to children’s drawings.

The pipeline they propose for animating human drawings consists of four steps and is given in Figure 2.2. As the first step, they propose to use a convolutional neural network, i.e., a Mask R-CNN, to detect the human character in an RGB input image (Figure 2.2(a)). In the second step, based on a bounding box detection (Figure 2.2(b)) they apply classical computer vision algorithms to segment the character from the background. The approach for this step can be summarised as follows, based on the bounding box the input is

2. RELATED WORK

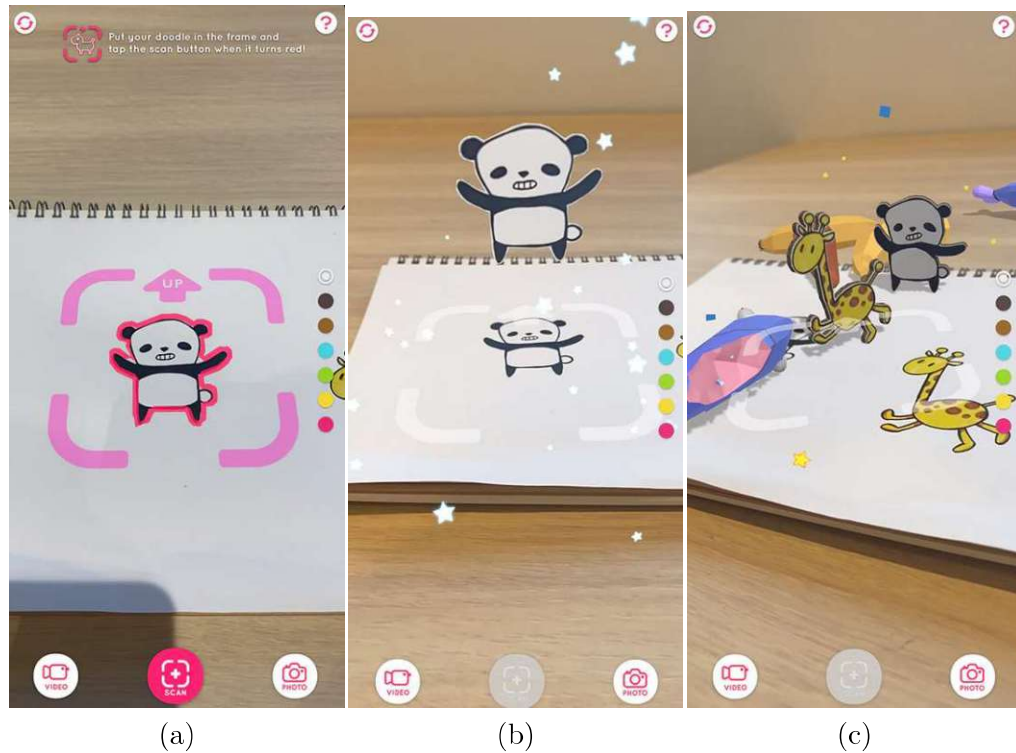


Figure 2.1: (a) A hand-drawn character is detected. (b) The detected character is brought to life as a textured 3D model in Augmented Reality. (c) A simple lively AR environment with multiple randomly animated characters [Jun].

cropped and converted into a grayscale image. On the resulting image, they apply adaptive thresholding followed by morphological closing and dilating that removes the noise of the mask. In the last step, they use a flood fill algorithm to ensure closed groups of foreground pixels. After that, they extract polygons based on the foreground pixels and proceed to process the one with the largest area. In the third step, a novel convolutional neural network is used to estimate the pose of the character. They found previously researched motion models to not perform well on drawn human figures due to the wide variety of appearances. The output of the network is an individual heatmap for each joint location of the pre-defined human skeleton used for animation. Using a simple search for the maximum value of each heatmap they assign the joint to that position shown in Figure 2.2(c). Based on the matched joints the motion, i.e., based on motion capture, the animation is transferred to the drawn human character as shown in Figure 2.2(d).

The proposed pipeline focuses solely on retargeting human motion to hand-drawn characters representing humans. We propose a fully automated pipeline computing an animation of a hand-drawn character based on semantic information of the hand-drawn character. Specifically, a humanoid character should be retargeted to a humanoid motion, while i.e., quadruped characters are retargeted to a quadruped motion.

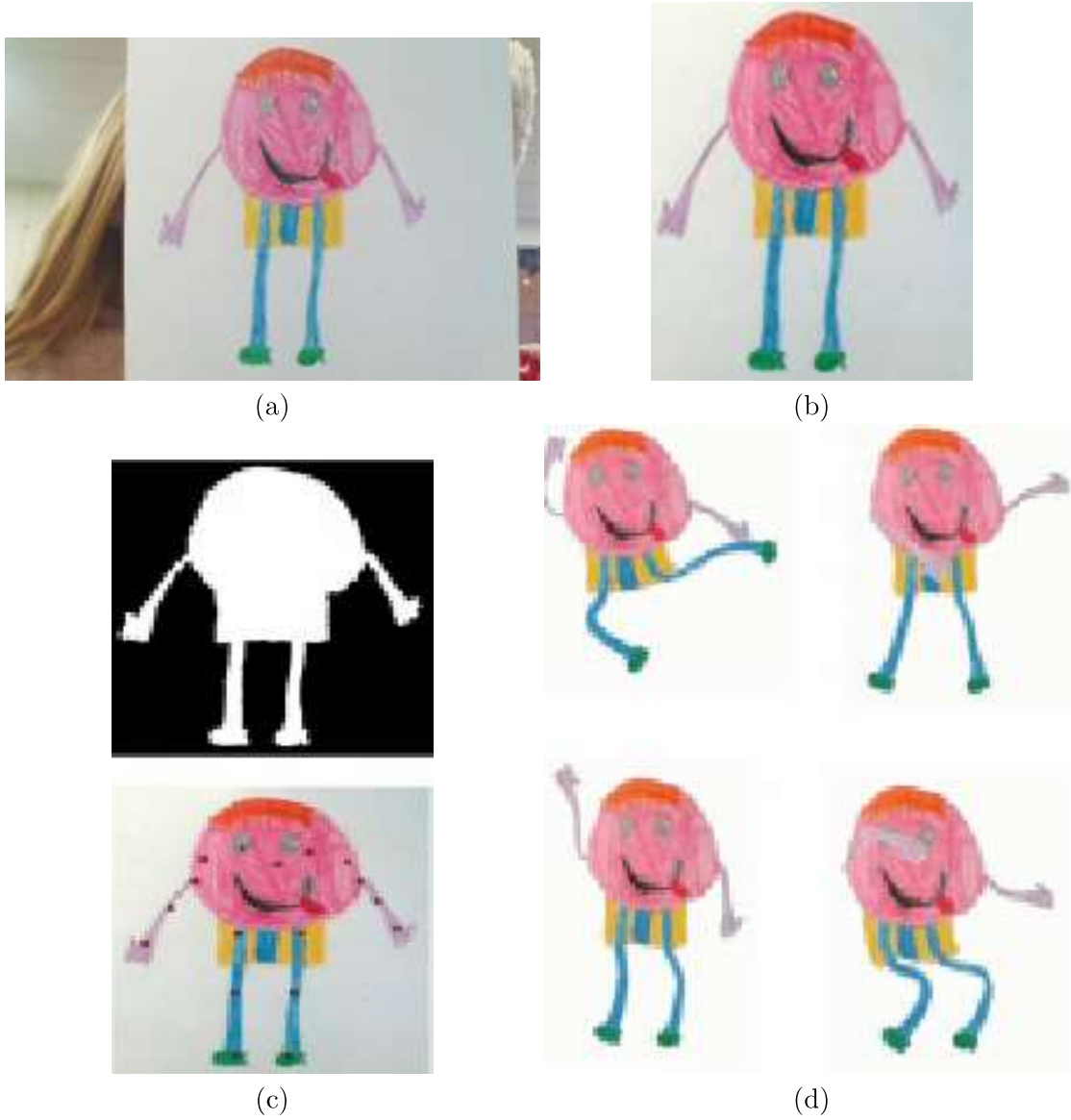


Figure 2.2: (a) RGB Image as the input of the pipeline. (b) The detected hand-drawn character. (c) The segmentation mask separates the character from the background and the estimated joint key points (annotated in red). (d) Different animations applied to the hand-drawn character [SZL⁺23].

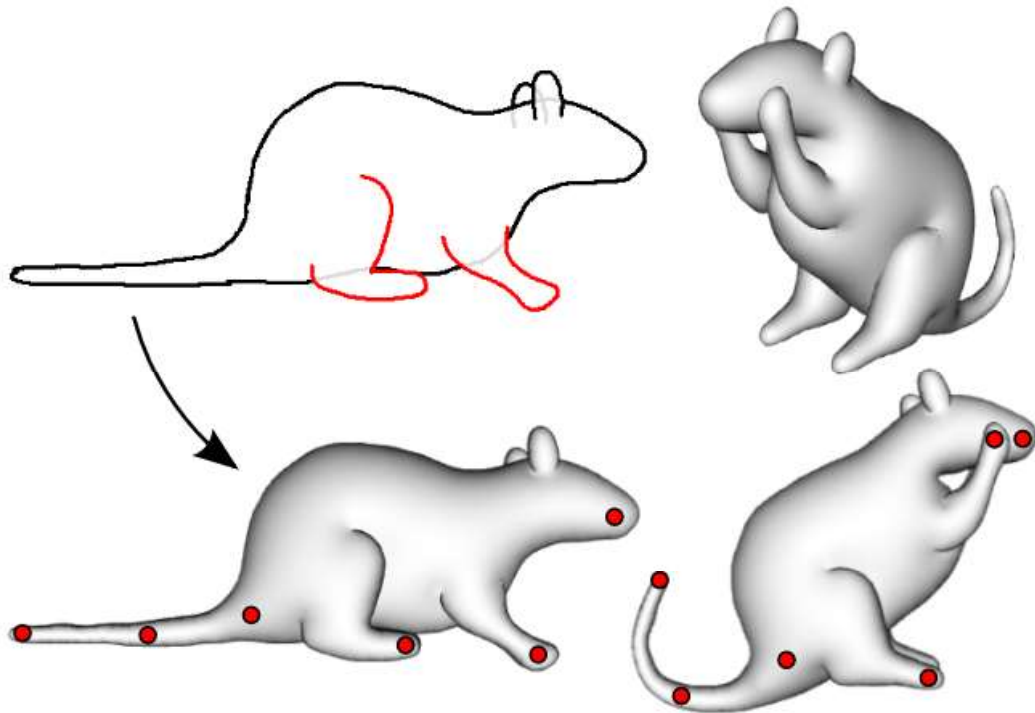


Figure 2.3: Example of a sketch (top left) transformed in a 3D model and animated by dragging control points (red) with the mouse. [DSC⁺20].

Dvorožňák et al. [DSC⁺20] propose a novel framework, Monster Mash, that allows users to create 3D models by hand-drawing characters in 2D. Users draw characters digitally specifying information for each stroke. Three types of information can be specified:

- The stroke lies behind another stroke, marked grey in Figure 2.3.
- The stroke lies in front of another stroke, marked black in Figure 2.3.
- The stroke should be interpreted as a symmetric part, meaning the given part, i.e., a leg, will be replicated twice by the pipeline, marked in red in Figure 2.3

Using a novel rigidity-preserving, layered deformation model, which in combination with a 3D inflation algorithm produces smooth 3D models. Further, once the model is created, users can animate the 3D character by simply dragging control points, marked in red in Figure 2.3, into different positions to move parts of the character associated with the control point.

The approach by Dvorožňák et al. [DSC⁺20] relies on the user input to be done on hardware, such as a tablet or computer, to classify each stroke into one of three categories.

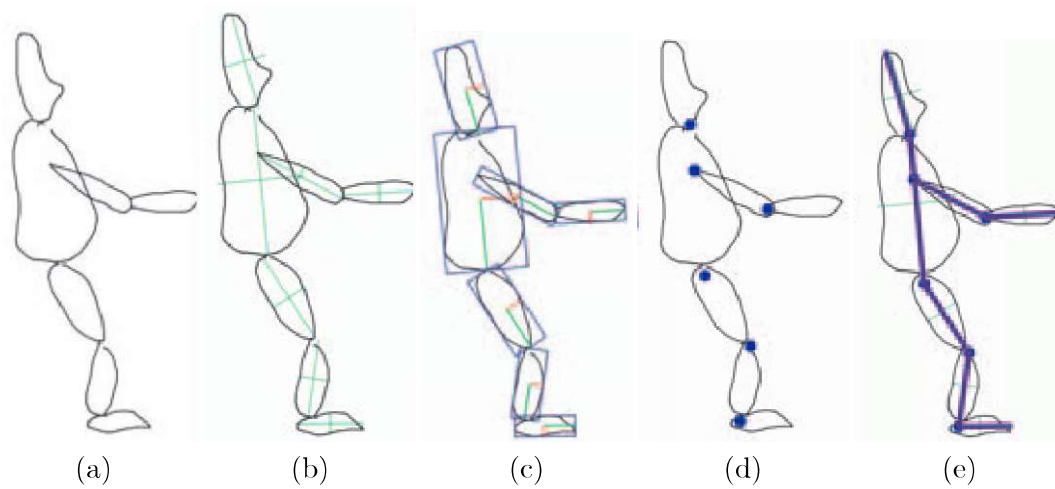


Figure 2.4: (a) The hand-drawn character consisting of seven body parts. (b) Computed major and minor axes of each part. (c) Oriented bounding boxes of each part. (d) Computed joint locations between the parts marked in blue. (e) Final computed skeleton with end-effectors inserted. [TBVDP04]

We propose that the input is created but not limited to a physical drawing, which lowers the barrier for users. Another differentiation is that they allow the user to animate the generated character whereas, we propose to automatically define the animation for the given input. This alleviates the user of the tedious task of creating the animation. As for users new to animation, it is an especially hard task to get the expected result.

Thorne et al. [TBVDP04] present a system allowing users to create an animation for a hand-drawn character. A character is expected to be drawn sideways and consists of exactly seven parts, the head, torso, upper arm, lower arm, upper leg, lower leg and foot, which are shown in Figure 2.4(a).

Based on the given input the system computes the major and minor axes of each of the seven body parts, shown in green in Figure 2.4(b). In the second step, the oriented bounding boxes are computed for each body part, illustrated with blue boxes in Figure 2.4(c). Based on the bounding boxes, joint locations are computed, shown as blue dots in Figure 2.4(d). In the last step, the computed joints are used to define the skeleton, by adding a bone between the joints. Additional end-effectors are added for the top of the head and similarly at the end of the foot as well as the tip of the arms. These end-effectors are connected with the rest of the skeleton as shown in Figure 2.4(e). In the last step, the legs and arms are duplicated on the other side of the character.

Once the character has been created by the system, users can sketch motion to animate the character. The sketched motion shown in Figure 2.5(a) is tokenized, meaning the sketched motion is split into multiple parts, and then classified depending on the properties of the token, i.e., the steepness of the segment. Based on the tokens, motion is synthesized by employing a parametrized key-frame-based motion synthesis algorithm. Figure 2.5(b)

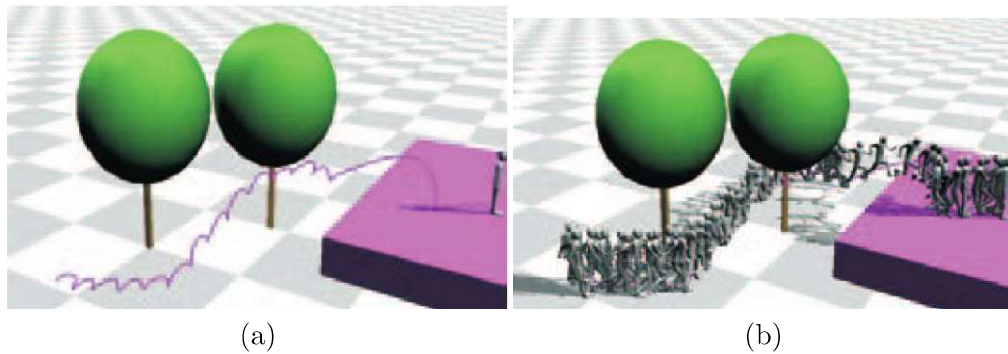


Figure 2.5: (a) Motion sketched for walking around two trees and leaping onto a platform. (b) Snapshots of the created animation for the character [TBVDP04].

shows snapshots of the character animation based on the sketched motion.

The system focuses on a defined character of exactly seven parts while we focus on the identification of different motion types for an unknown number of different parts. Due to the pre-known number of joints their skeleton generation is straightforward, while in this thesis the skeleton of the characters might differ a lot from the reference skeletons.

Jain et al. [JSMH12] take a different approach to the previously discussed related work. The basic idea is to generate a virtual 3D proxy, which is an approximate representation in 3D using cylinders that match the area of the 2D character. This allows users to enhance 2D animated characters by utilizing physical simulation, e.g., to create clothes for the hand-animated character. Further, their system allows traditionally trained animators working in 2D to influence the performance of a 3D character by hand-drawing 2D animations.

In the first step, the user is asked to annotate additional information of the hand-drawn character. The user annotates the joint positions by marking them with coloured dots as shown in Figure 2.6(a) on a frontal view of the skeleton to avoid any occlusions. Further, the user segments the inner body parts by colouring the inside of the hand-drawn character, which can be seen in Figure 2.6(b) matching the segmented parts with the associated joints. Lastly, the user annotates the bounding boxes of the different body parts as shown in Figure 2.6(c), where the bounding boxes of the right leg have been omitted for clarity, by drawing the approximate bounding boxes.

In the second step, the user is asked to select a motion capture sequence, from a large motion capture database, that has a similar motion sequence as the hand-drawn character. This is necessary to augment the data with depth information that resolves the depth ambiguity of the 2D animated sequence. Another necessary user input is an estimated camera to resolve the ambiguity of whether the camera is moving around the character or the character is moving on its own. Based on the user-estimated camera, the system can resolve the camera-character-motion-ambiguity.



Figure 2.6: (a) Coloured dots annotate the position of the joints of the hand-drawn character by the user. (b) Segmentation of the body parts by the user. (c) Hand-drawn approximation of the bounding boxes by the user [JSMH12].



Figure 2.7: Different frames of the hand-drawn character augmented with cloth using the information of the virtual 3D proxy [JSMH12].

The character is then approximated by 3D cylinders that track the lines of drawings inside each of the bounding boxes annotated by the user, where the radius of the cylinder matches the size of the character parts. In the last step, the viewpoint needs to be approximated by the system to maintain the naturalness of the sketched motion. It is also necessary to preserve ground contacts of the feet while transferring the style of the hand-drawn character animation onto the 3D proxy, which can be seen in the first frame shown in Figure 2.7 where the cloth needs to touch interact with the ground where the feet touch it.

Figure 2.7 shows the end result of their system, which allows augmenting the hand-drawn 2D animated character using the computed proxy. Another application of the pipeline can be transferring the stylised motion, drawn by the user in 2D, to 3D characters allowing users to create 3D animation using 2D drawings.

We focus on less user interaction especially regarding the animation of the character compared to the approach presented by Jain et al. [JSMH12], which approach requires a minimum expertise of the user to be able to create the animation. Further, they require a large database of motion capture data, that is hard to obtain for casual users, which we do not rely on.

We summarise the contributions of this thesis beyond the state-of-the-art in the following. While some previous work regarding retargeting motion to hand-drawn characters focuses on mapping to a single type of motion, i.e., humanoid, we focus on using semantic information of the drawings to retarget different types of motion based on the given character. Other previous work focuses on professional users with background knowledge about animation, whereas our approach is targeted also at novice users by providing a pipeline that is fully automatic without the need for manual input to alleviate the user of tedious animation work.

Technical Background and Definitions

This chapter discusses the technical background of the thesis. Key concepts and definitions are defined that help to understand the following chapters.

3.1 Machine Learning, Neural Networks, and Deep Learning

Following the definition of Mitchell et al. [Mit07] learning with regards to computer programs is defined as learning from an experience E with respect to a task T and a performance measure P , if the performance of the program at tasks in T , as measured by P , improves with experience E .

As a simple and popular example of machine learning, we can define T as playing Go, a popular board game, define the performance measure P as the percentage of games won and lastly the experience E as playing games against the program itself. In certain tasks, such as games like Go and Chess, a machine learning program can excel the performance of a human. AlphaGo, a program presented by Silver et al. [SHM⁺16], shows that it is possible to defeat the human Go champion and even excel with a later generation of AlphaGo being able to defeat the previous one with 100 wins to zero losses [SSS⁺17].

According to Bishop and Nasrabadi [BN06] machine learning can be split in the following general categories:

- **Supervised** learning describes a learning strategy that uses a dataset of example input as well as the target value for each input example. The dataset consists of individual data points that are described by their features $x \in X$ and the target

values $y \in Y$. The goal of supervised learning is that after learning it is possible to predict the correct target value y based on the given features x . An example of the goal of supervised learning can be a classification task, such as classifying a handwritten digit based on the MNIST dataset [Den12].

- **Unsupervised** learning uses a dataset of example input similar to supervised learning. While supervised learning relies on known target values $y \in Y$, unsupervised learning does not rely on the target values for each input example. The goal in such problems is to find groups inside the data, called *clustering*. Another example is to determine the distribution of the data, which is called *density estimation*.
- **Reinforcement** learning differs vastly from the two previous learning methods. The machine learning model learns by repeating a task inside a dynamic environment on the basis of trial and error. The machine learning model performs an action and is then rewarded, which *reinforces* correct actions, and observes the changes in the environment. The aforementioned machine learning model AlphaGo [SHM⁺16] is an example of reinforcement learning. No example input is provided to the machine learning model. Instead a sequence of states and actions, i.e., placing a stone, in which the model can interact with its environment, i.e., the Go board with the stones of the players.

In recent years significant advances have been made in the field of machine learning with the introduction of Artificial Neural Networks (ANN). Figure 3.1 shows a typical structure of a feedforward neural network (FNN), which is an ANN.

The FNN consists of three separate components shown by red boxes in Figure 3.1. The components of an ANN are usually referred to as *layers*, where data is passed from one layer to the next until the final output is calculated. Each layer consists of one or multiple sub-components called *neurons*. A neuron is an atomic unit of a neural network. It performs one operation based on the input and forwards the result of the operation to the next layer. The operation of a neuron is described with Equation 3.1 where it receives the input vector X consisting of n elements denoted as $X = \{x_1, x_2, \dots, x_n\}$. Each of the input values is multiplied with the respective weight w_i and all of the products are summed. The weights for each neuron represent the parameters of the network that are *learned* during the training of the network. Additionally Equation 3.1 shows that an additional value b , representing the bias, is added to the sum of products. Tom Mitchel [Lea97] describes the bias as a means to control the behaviour of a layer without changing the input values.

$$y = \mathcal{F}\left(\sum_{i=1}^n (x_i \cdot w_i) + b\right) \quad (3.1)$$

John Kelleher [Kel19] notes that many relationships in the world that we want to model are nonlinear and attempting to model these relationships using a linear model, that

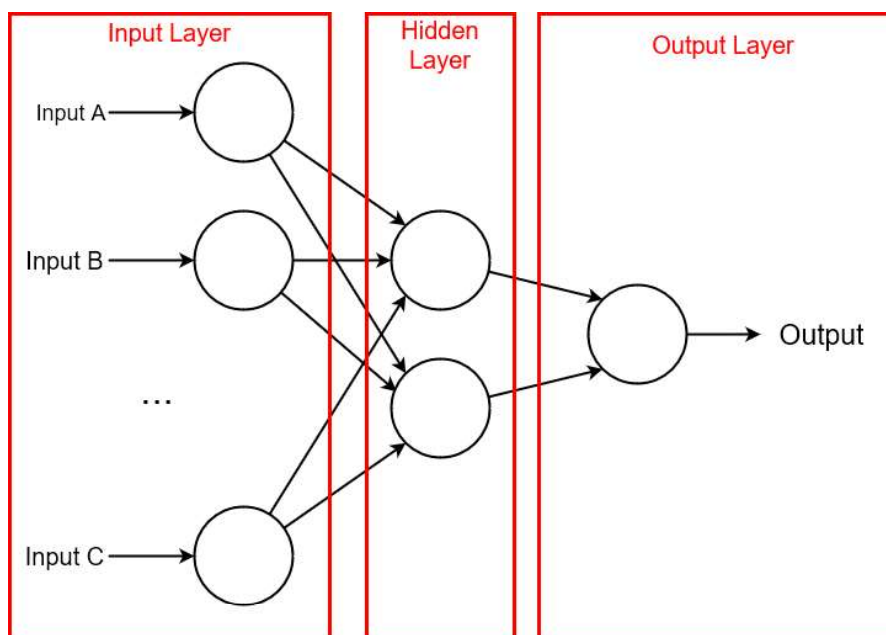


Figure 3.1: Basic structure of an Artificial Neural Network adapted from Keiron O’Shea and Ryan Nash [ON15].

consists of linear operations, makes the model very inaccurate. Therefore, the function \mathcal{F} , in Equation 3.1, is performed by each neuron which is referred to as an activation function. These functions allow to introduce non-linearity to the neural network. One popular example of such an activation function is the rectified linear unit (ReLU). Its mathematical definition can be seen in Equation 3.2 first introduced by Kunihiko Fukushima [Fuk75]. Other activation functions include the *sigmoid* function defined in Equation 3.3, *hyperbolic tangent* as shown in Equation 3.4 and *softmax* as defined in Equation 3.5. The softmax activation is of special interest, as it can be found in many different networks in the last layer when calculating the output.

$$\mathcal{F}(x) = \begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases} \quad (3.2)$$

$$\mathcal{F}(x) = \frac{e^x}{e^x + 1} \quad (3.3)$$

$$\mathcal{F}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (3.4)$$

$$\mathcal{F}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}} \quad (3.5)$$

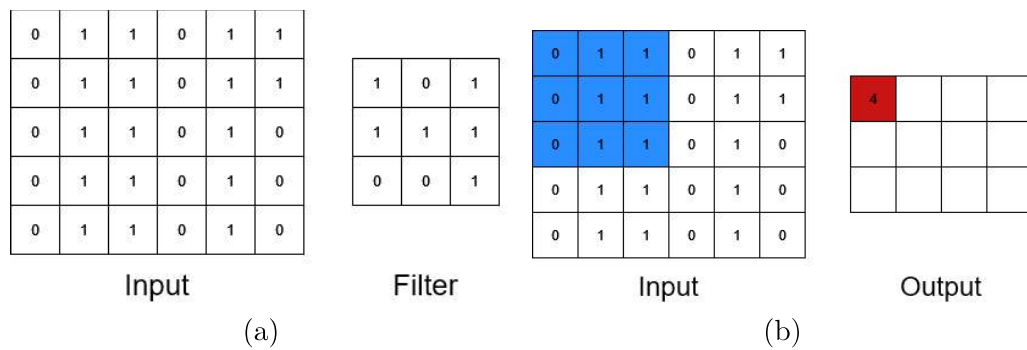


Figure 3.2: (a) The two-dimensional input and the filter used in the convolution operation. (b) The filter is then moved step-by-step over the input, for example, the filter is overlaid with the input marked in blue for the first step and the calculated results are written to the output marked in red, to compute the results of the convolution operation. Adapted from Introduction to Convolutional Neural Networks [Ser23].

$$S_x = \begin{pmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{pmatrix} S_y = \begin{pmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} \quad (3.6)$$

Deep Neural Network refers to an ANN with multiple hidden layers. A more specific neural network with particular relevance to the field of computer vision is the **Convolutional Neural Network** (CNN), which uses multiple hidden layers belongs to the area of deep learning [AMAZ17]. CNNs offer the advantage of reducing the number of parameters that are needed when working with difficult tasks, such as image data where applying convolutions reduces the size of the data between two layers as shown in Figure 3.2(b). The layers of a CNN can be summarised as the following four basic layers:

- **Convolutional Layer** implements the convolution operation, which is the application of an $N \times N$ sized filter using a sliding window on the input data. An example of a convolution can be seen in Figure 3.2(a) where a two-dimensional input and a two-dimensional filter are defined. By sliding the filter over the input image the output of the convolution is computed, the first step is shown in blue in Figure 3.2(b), the output can be computed shown in red in Figure 3.2(b), by element-wise multiplication of the values of the filters and the values in the input at the current position of the filter. Those values are then summed up and defined as the output of one convolution operation. The convolution reduces the resolution of the input matrix. To avoid loss of information outer values of the input *padding* can be employed, which is a technique that adds additional values at the border of the input. There are different approaches to how the values are assigned, one example can be to duplicate the border values themselves. Another important parameter is the *stride*. The stride defines how much the filter moves between consecutive

applications when sliding the filter over the input. In the given example a step size of two would reduce the output resolution without padding to three-by-three. There are many different ways to define the values of a filter as well as the size of the filter and as such filters have been of high interest in previous research. A widely known filter that is used for example is the Sobel operator, which are two three-by-three filters as defined in Equation 3.6. The Sobel operator is used to detect edges in an input image. The output of the convolutional layer is often referred to as a feature map. For example, using the Sobel operator as filters the output of the layer contains the edge features of the input.

- **Pooling Layer** whose main purpose is to reduce the computational complexity by decreasing the dimension of the input. There are different operations available, such as *max pooling* which takes the maximum value of the window overlaid with the input, as shown in Figure 3.3, and sets it as the output. The window is defined by its size and stride when applying it to the input. There are many different operations possible besides max pooling, such as average pooling, which calculates the average value, minimum pooling, calculating the minimum value, and others.
- **Fully Connected Layer** is a layer, where each neuron is connected to each neuron in the next layer. This means that every value of the inputs influences every value of the output. The main purpose of the fully connected layer is to combine all outputs of the previous layers, represent local information, to identify larger patterns. For classification tasks, the last fully connected layer combines the features to classify the images, therefore the number of outputs of the fully connected layer is the number of classes to identify.
- **Dropout Layer** sets input elements to zero with a given probability. During the training of the network, the layer randomly sets inputs to zero to avoid overfitting the CNN as well as reducing the number of parameters, which decreases the inference time of the network.

In recent times specialised architectures for neural networks have been developed for specific tasks. For our proposed pipeline, specifically for the tasks of detection and instance segmentation, a popular architecture is Mask Region-based Convolutional Networks (Mask R-CNN) introduced by He et al. [HGDG17]. It is based on Fast Region-based Convolutional Network method (Fast R-CNN) proposed by Ross Girshick [Gir15], which excels at object detection.

The Fast R-CNN consists of two stages which can be referred to as convolutional backbone architecture and the network head. First, the whole image is processed by a convolutional neural network, consisting of multiple convolutional layers as well as a max pooling layer, to compute the feature map as seen in Figure 3.4. Further, a region proposal network proposes object candidates as regions of interest (RoI). An example of an RoI is shown as a bounding box in red in Figure 3.4. The RoIs are then projected onto the feature map and passed to the second stage.

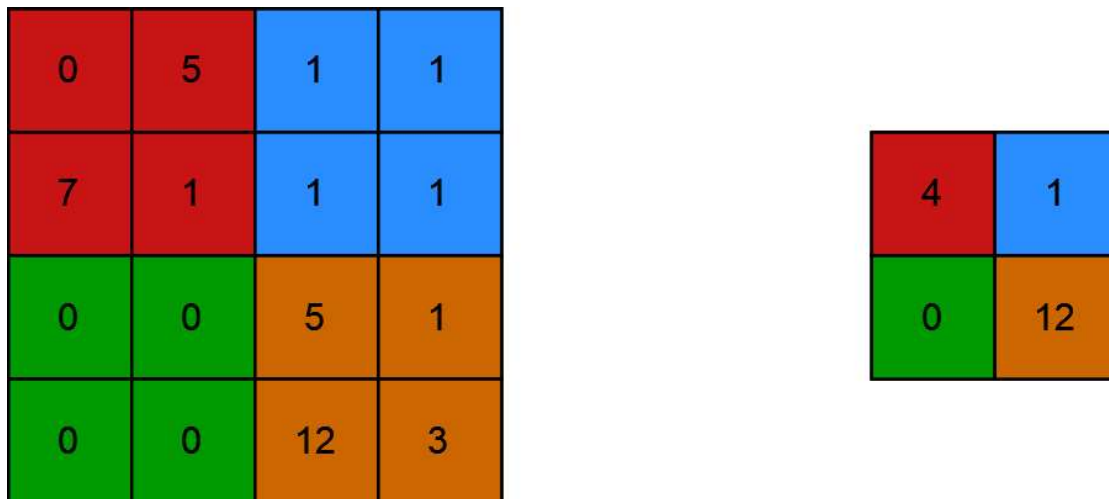


Figure 3.3: Max pooling operation on the given input produces the output on the right side, for a window with a stride of two and a size of two-by-two. The colours correspond to the window overlaid with the input and the output.

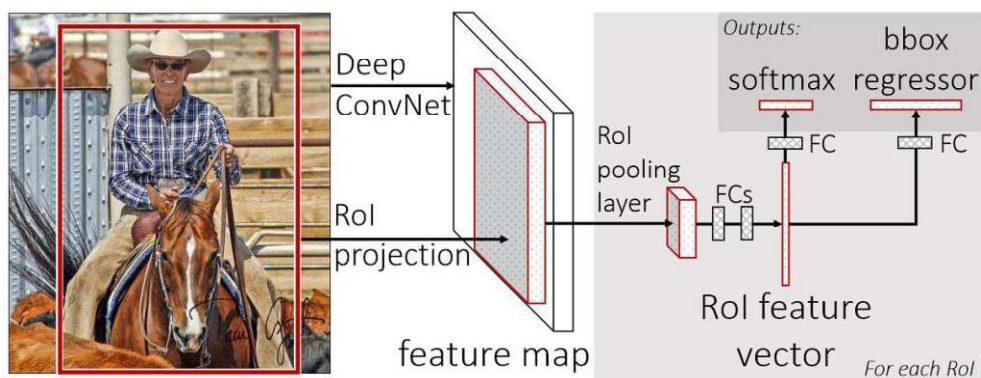


Figure 3.4: Basic architecture of the Fast R-CNN architecture as proposed by Ross Girshick [Gir15].

For each object proposal, the RoI pooling layer computes a fixed-length feature vector based on the feature map. These fixed-length vectors are then passed into multiple fully connected layers which compute the RoI feature vector. This vector is then passed to the output layers.

The first output is defined as the classification of the object. A fully connected layer is used to reduce the dimension of the RoI feature vector to the number of classes that the network can differentiate. The resulting vector is passed to a softmax layer that computes the probability of the object belonging to each of the N defined classes. The second output is defined as the exact bounding box positions encoded as four numbers. This means the network output gives the type of object and its position in a given image.

3.2 Graph Theory

Our proposed pipeline depends on the skeleton of the character to be represented as a graph, therefore we describe essential definitions and algorithms that are used throughout the following chapters. Graph theory has been a popular and highly researched area for many years. Graphs have various applications in other fields outside of pure mathematics. A Graph is typically defined by a set of vertices or also called nodes and a set of edges. Each edge defines the relationship between two vertices. Therefore, let the Graph $G = (V, E)$ be defined by a finite set of vertices V and a finite set of edges E . While it is, in general, possible for the sets of V and E to be infinite, not all algorithms can be applied to graphs with an infinite number of vertices or edges, so for simplicity and relevance to this thesis, every following graph is assumed to be finite.

Let an edge $e \in E$ be defined as an unordered tuple of two vertices, such that $e = (v, u)$ with $v, u \in V$. Graphs defined this way are called simple *undirected* graphs, as the edge does not imply any directedness. Figure 3.5 shows a undirected graph $G = (V, E)$ with $V = \{a, b, c\}$ and $E = \{e_1, e_2, e_3\}$. A *loop* is defined as an edge that connects the same vertices, such that $e = (v, v)$ with $v \in V$.

Contrary to undirected graphs there is the possibility to also encode directedness in edges such that the graph $G_D = (V, E)$ is defined as a set of vertices V and a set of edges E defined as $e \in E$ with $e = \langle u, v \rangle$. $\langle u, v \rangle$ represents an ordered tuple such that $\langle u, v \rangle \neq \langle v, u \rangle$. A graph defined in this way is called a *directed* graph.

There are many more possible ways to define graphs, e.g. graphs allowing or a vertex pair to be connected by multiple edges are usually referred to as *multigraph*.

The vertices of a graph have different attributes, such as their degree $deg(u) = n$ with u being a vertex of a graph and n being the number of vertices v that are connected to u with an edge $e = (u, v)$. If an edge exists between u and v the vertices are referred to as neighbours. An edge $e = (u, v)$ is called an *incident* edge for the vertices u and v . The degree of vertex a in Figure 3.5 is two, as it has two neighbouring vertices b and c .

An alternative way of describing a graph is to define it using an *Adjacency Matrix*. As seen in Figure 3.6, each row and column defines a vertex $v \in V$ and every edge $e \in E$ is

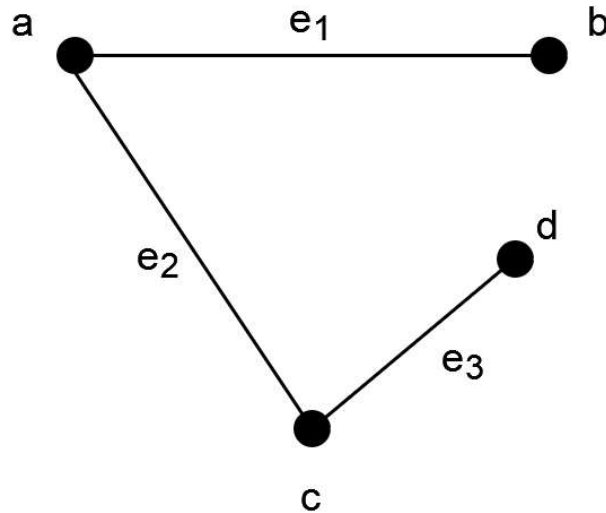


Figure 3.5: A simple undirected graph G represented by a node-link diagram. Each vertex is represented as a circle (node) and each edge is represented by a line (link) linking the two respective vertices.

	a	b	c	d
a	0	1	1	0
b	1	0	0	0
c	1	0	0	1
d	0	0	1	0

Figure 3.6: The adjacency matrix representation of the graph G from Figure 3.5.

represented as a 1 in the matrix. The entries correspond to the graph G as shown in Figure 3.5. The adjacency matrix is a different representation of graphs, compared to a graph being represented as a node-link diagram as seen in Figure 3.5, and can be used for many different algorithms in optimisation problems.

Graphs are an important topic in a very diverse research field ranging from applications in computer science to electrical engineering. There are many different graph algorithms such as *traversal* algorithms, *shortest path* algorithms and *minimum spanning tree* algorithms to name but a few algorithms.

For example, graphs can be used to model and analyse social networks. Clustering algorithms can help understand connections, influence and patterns in social communities. Another application of graph algorithms is in circuit design and analysis, where vertices represent components and edges represent connections between the components. An important algorithm was introduced by Kerningham and Lin [KL70] which partitions the graph and minimises the number of interconnections between electric components to

reduce the cost of electrical components.

An important role of graphs and their algorithms is in modelling transportation systems such as road networks. A typical application of traversal algorithms is navigation in road networks. In bioinformatics graphs can be used to represent protein interactions, genetic networks, and metabolic pathways such as Metabopolis presented by Wu et al. [WNSV19].

Another important category of graph algorithms with special relevance to this thesis are graph matching algorithms. Graph matching algorithms aim to find correspondences between two or more vertices or edges of different graphs. An example of a graph-matching algorithm is the *graph edit distance* presented by Horst Bunke [Bun97]. The graph edit distance measures the dissimilarity between two graphs G_A and G_B . The difference can be expressed by the minimum number of edit operations to transform the graph $G_A = (V_A, E_A)$ into the graph $G_B = (V_B, E_B)$. The edit operations f are defined as follows:

- **Vertex deletion** means deleting a vertex $v \in V_A$, which also requires removing all edges e where v is part of
- **Vertex insertion** means inserting a new vertex $v \notin V_A$ into V_A
- **Vertex substitution** means replacing a vertex $u \in V_A$ with a vertex $f(u) \in V_B$
- **Edge deletion** means removing an edge $e = (u, v) \in E_A$ without removing the vertices u and v
- **Edge insertion** means inserting a new edge $e = (u, v)$ with $u, v \in V_A$
- **Edge substitution** means substituting an edge $e = (u, v) \in E_A$ with an edge $e' = (f(u), f(v)) \in E_B$

Each of the edit operations can be assigned with a cost. In case each operation is assigned the same cost, the graph edit distance is defined as the minimum number of edit operations. If operations are assigned different costs, the graph edit distance is defined as the set of operations that has the minimum cost. Optimisation algorithms are used to find graph edit distance. Kaspar Riesen [Rie15] shows that the graph edit distance can be used in structural pattern recognition tasks, where the images that should be matched are represented as graphs based on extracted features.

3.3 Character Animation

Character Animation is an integral part of this thesis. It is the process of bringing fictional characters to life by creating movement, expression and emotions. Character animation is part of the larger topic of Computer Animation which describes the process

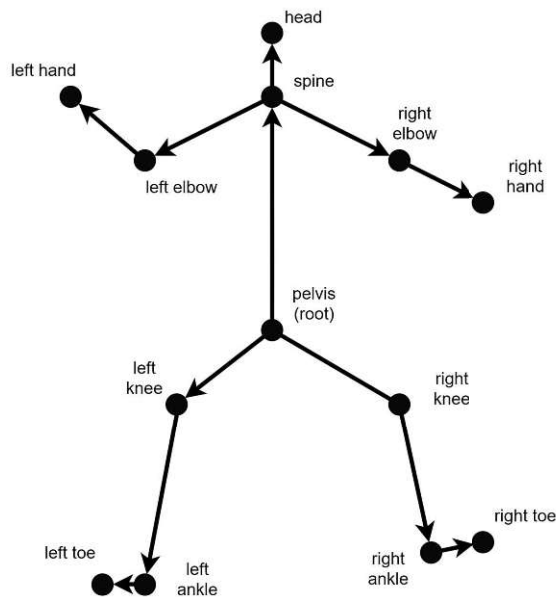


Figure 3.7: Simple skeleton representing a human character. Circles describe joints, while the directed links between circles represent the bones connecting the joints.

of using digital technology and computer software to bring scenes in movies and games to life. Especially focusing on games and movies [DKT98], character animation is relevant to create an immersive experience for users, among other factors such as sounds and visuals. Besides computer graphics computer animation is also relevant in other fields, such as robotics.

Character animation is a very diverse topic itself, with very complex sub-topics that are highly researched themselves, e.g. face animation specifically focusing on the animation of faces. The focus of this chapter is on the most relevant concepts and definitions with regard to character animation.

There are many different ways that a character animation can be modelled. One of the most prevalent methods is *skeleton-based* animation, often referred to as *Rigging*. It is a widely used technique in computer graphics for creating natural and realistic movements for characters and objects. For skeleton-based animation the basic idea is to represent it as a continuous deformation of a *skeleton* which is a hierarchical structure of joints that are connected by directed links [MHLC⁺22], as shown in Figure 3.7. Specifically, when used in character animation, the links connecting the joints are referred to as *bones*.

The base of each bone is the point where rotation is applied, changing the position of the joint attached to the tip of the bone. An example can be the rotation applied to the bone connecting the left elbow with the left hand in Figure 3.7 changing the position of the left hand. Joints that are only attached to one bone, at the tip, are referred to as *end-effectors*. Another special point is the *root* of the skeleton, which in Figure 3.7 is the

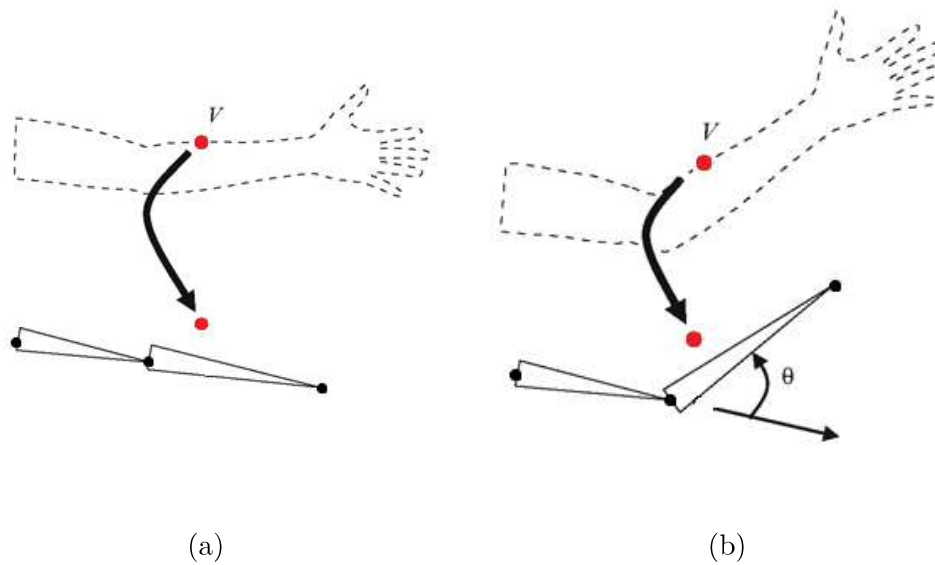


Figure 3.8: (a) A hand and two bones of the associated skeleton. Red marks a mesh-vertex of the mesh representing the hand and its corresponding position relative to the bone. (b) Applying a rotation θ to the bone connecting the hand end-effector shows the change of the mesh and associated mesh-vertices. Adapted from what-when-how [Pub].

pelvis joint, as it defines the position of the skeleton in the world space. The rotation of each bone, which determines the position of each joint, as well as the position of the root defines a unique *pose* of the skeleton.

Skeleton-based animation is most of the time used in combination with models that represent characters or other objects, which consist of one or multiple meshes. A mesh consists of mesh-vertices that are connected with mesh-edges where each closed loop of connected mesh-vertices represents a mesh-face.

The pose of the skeleton can be used to apply the pose to a mesh as shown in Figure 3.8 showing how the hand mesh changes when applying rotations to a bone associated with the hand. To achieve the correct deformation of mesh, a simple way is to assign a weight for each mesh-vertex that defines how much influence a certain bone has on that mesh-vertex. Figure 3.9 shows the influence of the bones on each mesh-vertex by colouring the mesh-vertices based on the colour of the bones.

The influence of the bones on each mesh-vertex ranges between zero and one. Mesh-vertices influenced by both bones have their colour mixed for example the vertices marked with p are influenced equally by both bones and therefore coloured in yellow. When rotating the green bone, it shows how the mesh deforms, meaning the positions of the mesh-vertices change, depending on how much influence the bone has on the individual mesh-vertex. Different techniques exist that solve the problem of assigning influence

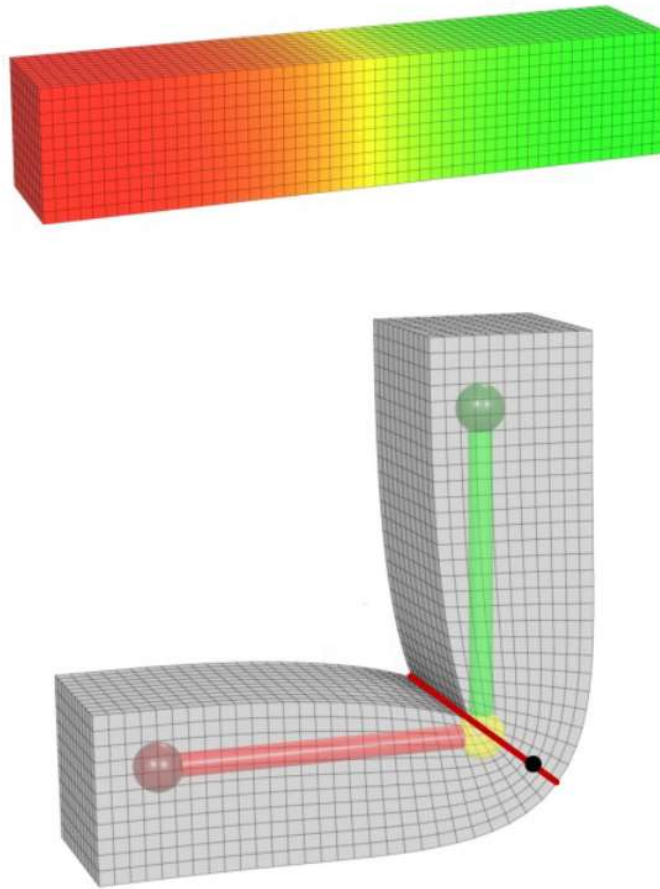


Figure 3.9: Skeleton-based deformation of a mesh based on two bones (red and green). The influence of the bones on each mesh-vertex is shown in the respective colour of the vertex [TP23].

of each bone to each vertex. One proposed approach is the multi-weight envelopment method described by Wang and Philips [WP02].

Algorithms dealing with the deformation of meshes using skeletons are referred to as *Vertex Skinning*. It is an actively researched topic according to Le Binh Huy and Deng Zhigang [LD12] with the focus of it being mostly on 3D models. Vertex skinning techniques are not limited to 3D characters and meshes but can also be applied to 2D characters as described by Pan and Zhang [PZ11], which are the focus of our proposed pipeline.

Inverse-Kinematics (IK) and **Forward-Kinematics** (FK) are research areas closely

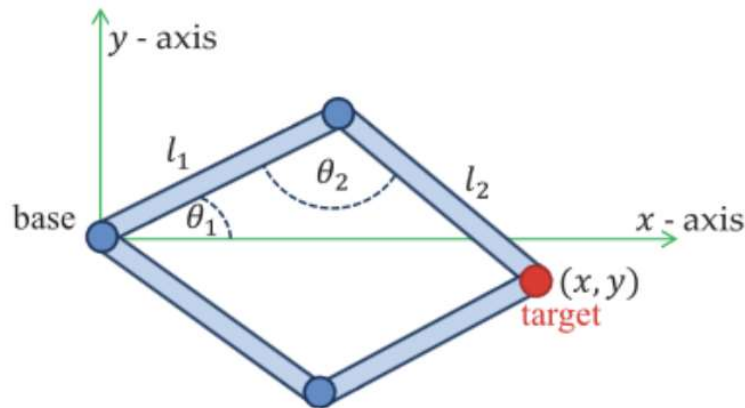


Figure 3.10: A problem with two possible solutions for Inverse Kinematics [ALCS18]. The red circle marks the end-effector while blue dots mark the joint and the root. Joints, base and end-effectors are connected by bones.

connected to vertex skinning and skeletal-based animation. For forward kinematics one must calculate the angles Θ_1 and Θ_2 representing the respective rotation of the bones l_1 and l_2 such that the end-effector, marked red in Figure 3.10, ends up on the coordinate (x, y) . This can be a very difficult task especially if a complex skeleton with many joints and bones is being used.

Using FK an animator needs to manipulate the rotation of different bones to place the end-effector to the desired position, which is a difficult task. IK allows an artist to manipulate the end-effector directly, i.g. placing the end-effector representing the toe of a character to a certain position. By solving a system of (kinematic) equations the necessary bone rotations are calculated in a way that the position is reached within the constraints (avoiding stretching of bones). Figure 3.10 illustrates the constraints of a simple skeleton in which the desired position of the end-effector leaves two possible solutions.

There are many different algorithms and many advances that can solve the problem shown in Figure 3.10 which can be found in the survey of Aristidou et al [ALCS18].

With IK and skeleton-based animation, an artist can create what is called a keyframe-based animation. This means for specified frames, the keyframes, the animator specifies the pose of the skeleton. All frames in between the keyframes are calculated by interpolating the pose of the previous and the next keyframe using IK. While keyframe-based animation is not limited in combination with skeleton-based animation it is a widely used combination of approaches.

The high degree of freedom when specifying a pose for a skeleton and the complex interaction of the mesh with its environment makes generating even short animations a complex and tedious task, especially if done by hand. Therefore the manual definition

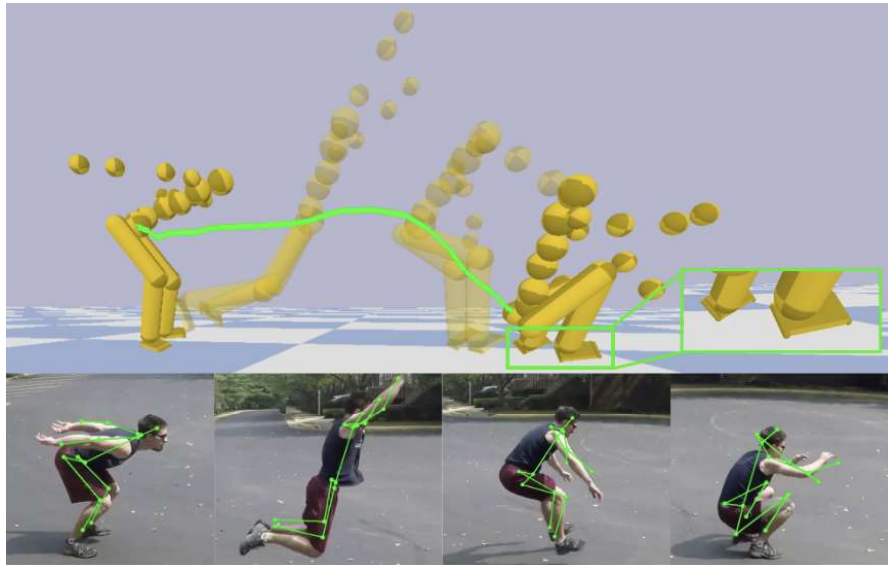


Figure 3.11: Markerless motion capture of a human jumping [SGXT20]

of the skeleton pose for each keyframe is not always feasible as it is very time intensive and requires a large knowledge of how, e.g. human motion works, to recreate a realistic movement.

The current state-of-the-art approach for creating life-like animations is by using the **Motion Capture** (MoCap) technique. The basic idea of MoCap is to automatically estimate the skeleton pose for each frame video sequence as shown in Figure 3.11. MoCap techniques can be split into the following categories:

- Sensor-based MoCap, uses sensors attached to an actor to track the position of the joints. E.g. electromagnetic sensors are attached and their positions are tracked through their interaction with an electromagnetic field generated by a base station.
- Optical MoCap, uses a camera to track the pose of the actor either by using reflective markers or using a markerless approach, such as presented by Shimada et al.[SGXT20].
- Hybrid MoCap, uses a combination of techniques to track the pose.

Figure 3.11 illustrates the idea of markerless monocular MoCap, a form of optical MoCap. From a sequence of images, the pose can be extracted using computer vision approaches, to define the keyframes of the animation. In recent years it has been especially driven by human-like animation according to Kitagawa and Windsor [KW20]. It alleviates the manual work of defining the skeleton for each keyframe, as those are usually retrieved in an automated way.

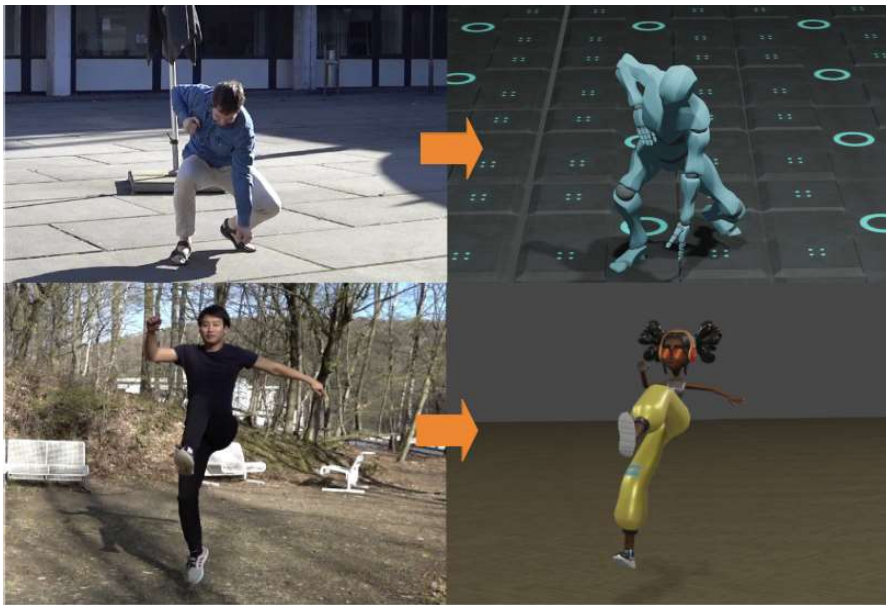


Figure 3.12: Retargeting of a pose captured by markerless motion capture [SGXT20]

Due to the high cost of creating high-quality motion captures recently a lot of research focused on combining CNN-based approaches with training data generated using MoCap. Holden et al. [HKS17] present an approach that allows to create life-like animations for unseen skeletons and environments. Characters in a game or movie should react naturally to the input of a user or artist and also react believably to other objects present in the scene, e.g. a human character walking up stairs. Starke et al. [SZKS19] show in their research that they are able to create specific animations for different characters and environments. The learned approach is necessary as not every situation can be cost-efficiently created using motion capture.

Zhang et al. [ZSKS18] shows that the previously mentioned learned approaches by Holden et al. [HKS17] and Starke et al. [SZKS19] are not limited to human or humanoid animation but can also be extended to the animation of animals. Quadruped motion, e.g. animation of a cat, can be learned in a similar way.

Another key concept, related to our pipeline for creating animation for hand-drawn characters, is the retargeting of motion. Cheung et al. [CBHK04] describe the main idea as a motion that is executed by one character and can be retargeted to a second character, such that the second character executes the motion. Figure 3.12 shows a motion generated by MoCap of a human that is retargeted to animate a virtual character. Motion retargeting allows to use one animation on any different character, making it possible to animate a large set of characters.

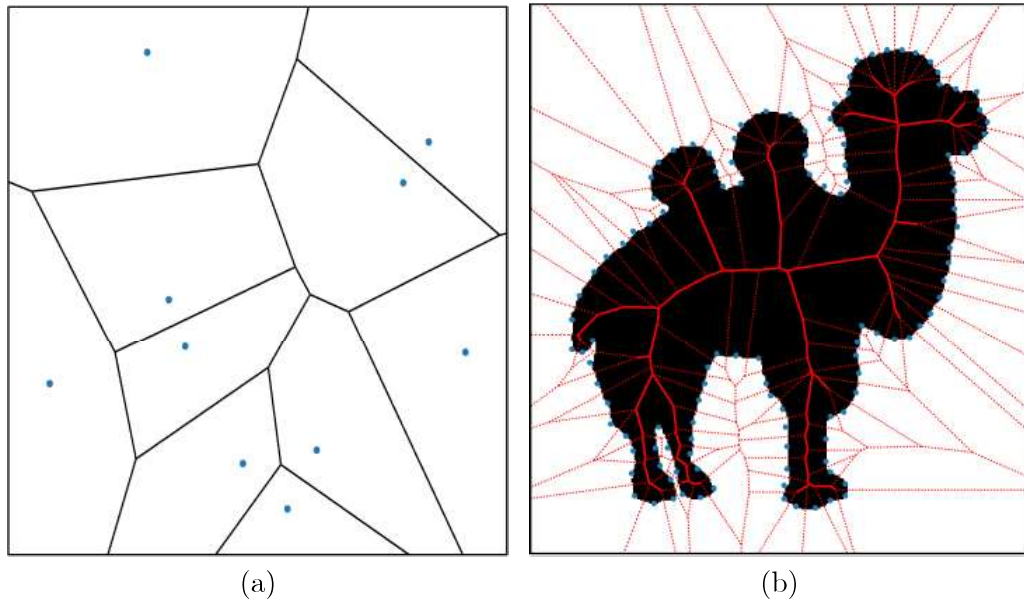


Figure 3.13: (a) Example of a Voronoi diagram on a set of points marked in blue. (b) Voronoi diagram of selected points on the contour (blue) of a shape [LGK19].

3.4 Creating Skeletons

Since our pipeline uses skeletons of unseen hand-drawn characters, an important related topic is the creation of skeletons. One popular approach for creating skeletons is using *Voronoi Skeletons*. For this, we first have to define what a Voronoi Diagram is. According to De Berg [DBVKOS97] the Voronoi diagram is given using Definition 3.4.1. Figure 3.13(a) illustrates a set of points P , marked in blue, separated into cells according to Definition 3.4.1.

Definition 3.4.1 (Voronoi Diagram). Let $P = \{p_1, p_2, \dots, p_n\}$ be a set of distinct points on a plane which are called *sites*. The Voronoi diagram of P is defined as the subdivision of the plane into n cells, one for each site in P , with the property that a point q lies in the cell corresponding to a site p_i if and only if $\text{dist}(q, p_i) < \text{dist}(q, p_j)$ for each $p_j \in P$ with $i \neq j$.

There are different algorithms for efficiently computing a Voronoi diagram such as the approach presented by Steven Fortune [For86]. As input, we define selected points of the contour of a shape, marked in blue in Figure 3.13(b). These points are used to calculate the Voronoi diagram. Max Langer [LGK19] refers to points, that are closest to two cells, as the Voronoi edges. Voronoi vertices refer to points connecting Voronoi edges and have more than three closest cells. Bold red lines in Figure 3.13(b) are all edges where generating sites do not lie next to each other on the contour. Using this property an initial skeleton can be defined.

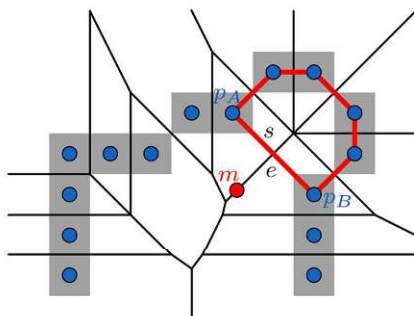


Figure 3.14: Grey boxes show the contour of the shape and black lines show Voronoi edges. Each pixel of the contour is used as a site. For each point m the length of the chord length is calculated, meaning the distance of the contour connecting the closest two sites p_A and p_B [LGK19].

The initial skeleton contains many spurious branches that we want to avoid. Robert Ogniewicz and Markus Ilg [OI92] introduce multiple measures of pruning the skeleton to avoid spurious branches that are introduced due to noise.

They propose to calculate a residual value for all of the Voronoi edge points. One way to compute the residual value is using the *Chord residual* as shown in Figure 3.14. The idea is, that for each Voronoi edge point m we calculate the length of the contour connecting the two sites closest to m , e.g. p_A and p_B . Once this value is calculated for each Voronoi edge point pruning the skeleton is simply applying a threshold and discarding all points that are below the threshold. After applying the threshold the Voronoi skeleton has been computed.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Methodology of Animating Hand-Drawn Characters

This chapter describes each of the five proposed pipeline steps. First, an overview is given of the pipeline and further sections explain the methodology of each step in more detail.

4.1 Overview of the Animation Pipeline

As shown in Figure 4.1(a), the input to the pipeline is a single image, which contains the hand-drawn character that should be animated. In the first step, the drawing is separated from the background of the image (Figure 4.1(b)).

In the second step, the drawing is semantically segmented, defining which parts of the hand-drawn character are limbs, body, etc. (Figure 4.1(c)). In the third step, a skeleton is generated where the joints and end-effectors are assigned a type based on the semantic information of the previous step, e.g. limb, body, (Figure 4.1(d)). This information is then used to identify the closest reference skeleton for the given skeleton (Figure 4.1(e)).

The final output (Figure 4.1(f)) is an animated mesh. In the following sections, each step will be described in detail.

4.2 Character Detection

The first step of the pipeline is to detect the hand-drawn character in the input image (Figure 4.1(b)). In this step of the pipeline, we assume the input is a single image that contains at least one hand-drawn characters that should be animated with the goal of separating them from the background for further processing.

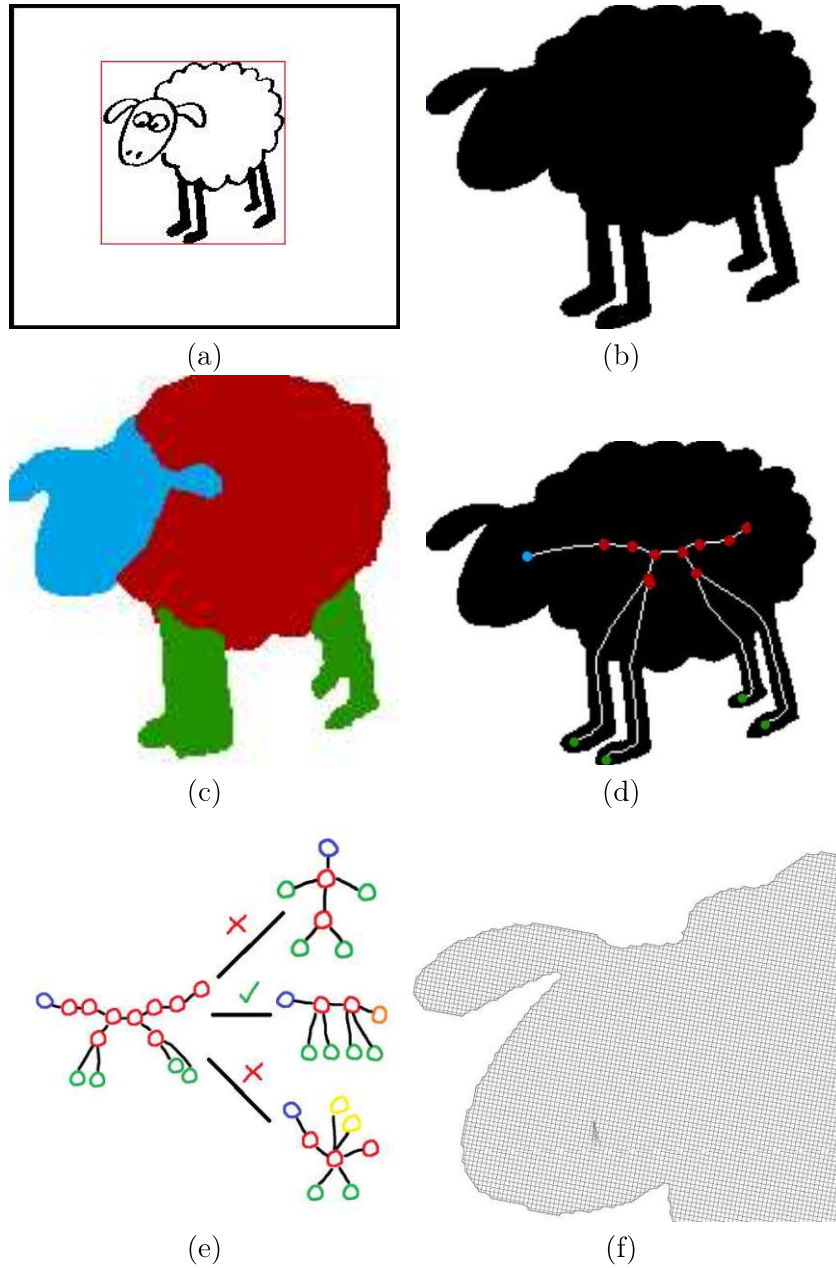


Figure 4.1: (a) Image containing hand-drawn character. (b) Detection step, where the character is separated from the background. (c) Semantic segmentation, where all parts of the sketch are identified. (d) Calculation of the skeleton. (e) Mapping of the skeleton to the best matching reference. (f) Triangulation of the sketch and rigging that results in the final animation.



Figure 4.2: Example input for the pipeline containing multiple hand-drawn characters.

Without loss of generality, we restrict the input image to a top-down view of the drawings to avoid distortions. Further, the drawing must be black foreground with white background to simplify the process. Alternatively, digital drawings can be used with the same aforementioned restrictions. Figure 4.2 shows an example of an input image captured from a top-down view into the pipeline with multiple hand-drawn characters.

Based on the input and the task of identifying the characters, it enables us to choose

from multiple different approaches. One option is to use computer vision approaches such as a contour-based approach presented by Suzuki and Abe [SA85]. Based on a binarised input image, which means that pixels of the image are either white or black, the contours of objects are computed. Contour refers to the outline or boundary of an object or shape in an image.

Another option is an edge-based approach as presented by John Canny [Can86]. The main idea is by applying a filter to an image we can compute the edges in the image. Using our assumptions on the input, we can find the edges between the black foreground drawing and the white background.

Both approaches come with some potential issues. For the contour-based approaches, we need to make certain assumptions, e.g. such that each contour describes exactly one hand-drawn character and inner contours are to be ignored. Another issue is if the lines of the drawings are not fully connected, but has gaps between the lines, we will find multiple contours. From these multiple contours, it is hard to determine whether they are different hand-drawn characters or whether all contours belong to the same character. While this problem can be solved by dilating the input, after thresholding, it only allows to close small holes. If we dilate the input too much it is possible that multiple characters will be detected as a single contour if their foreground merges.

Similar problems occur for the edge-based approaches for detecting characters. This means that it is difficult to differentiate a single from multiple hand-drawn characters if they are drawn close together. Further, if the drawings contain inner structures, as seen in Figure 4.2, it is hard to define a heuristic to compute the shape (black in Figure 4.1(b)) of the character robustly.

To avoid restricting the input further to make the above-mentioned approaches possible, we decide to use a machine learning-based approach. Smith et al. [SZL⁺23] show promising results for using CNNs to detect childrens drawings in an input image. Specifically, we choose the Mask R-CNN as proposed by He et al. [HGDG17] as it excels at object detection tasks.

The basic idea of the Mask R-CNN follows the architecture of the Fast R-CNN as presented in Chapter 3.1. Additionally to the bounding box and classification output of R-CNN, He et al. [HGDG17] propose to use an additional output for generating the exact mask of the detected object. Figure 4.3 shows the three outputs of the Mask R-CNN, which are a bounding box, the classification and the mask for each object.

For this thesis, the backbone of the network is the Residual Network (ResNet), a deep neural network introduced by He et al. [HZRS16]. The basic idea of ResNet is to avoid the problem of vanishing gradients for very deep neural networks. Residual building blocks are introduced to allow the network to learn fine-grained details as well as more abstract higher-level features at the same time.

Figure 4.4 shows a residual building block, where x denotes the input. While the input is passed through multiple layers a short-cut connection is made where the initial values of

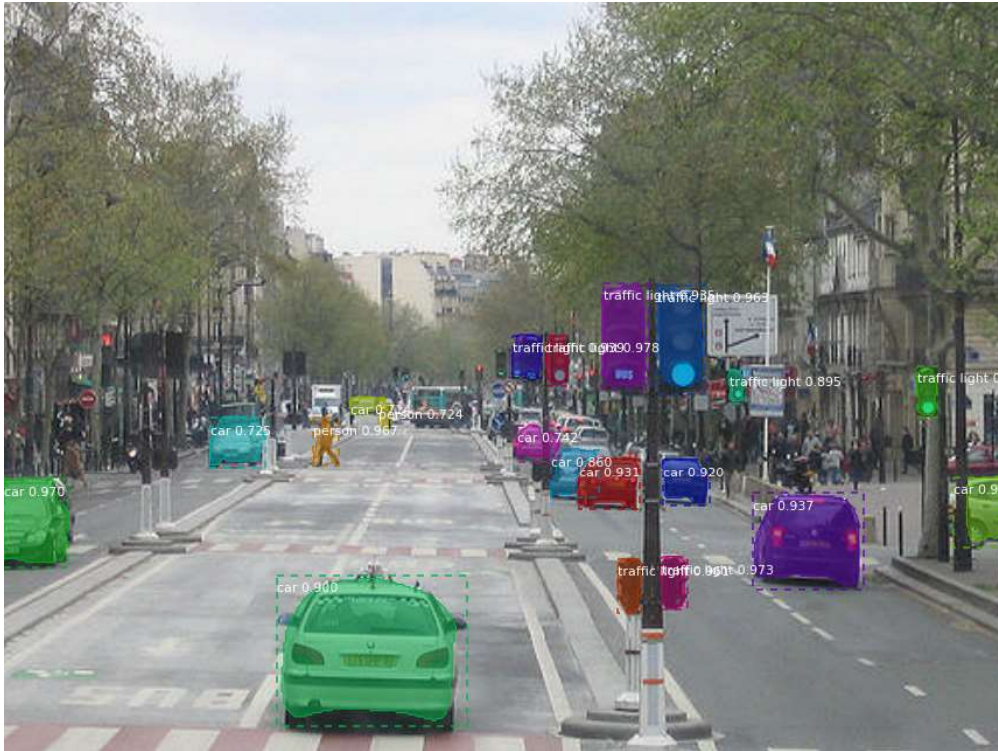


Figure 4.3: Output of the Mask R-CNN [HGDG17] showing bounding boxes, classification and mask for each detected object.

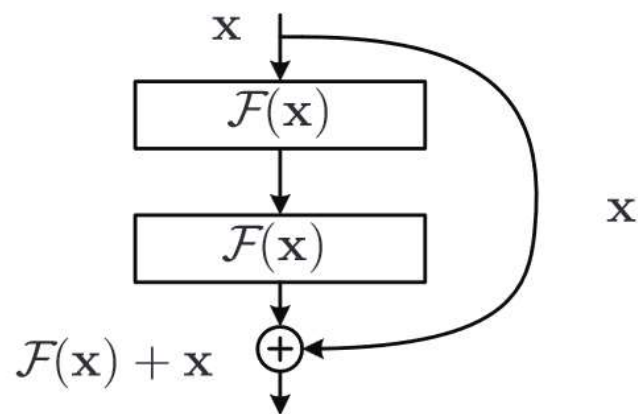


Figure 4.4: Residual learning building block adapted from He et al. [HZRS16].

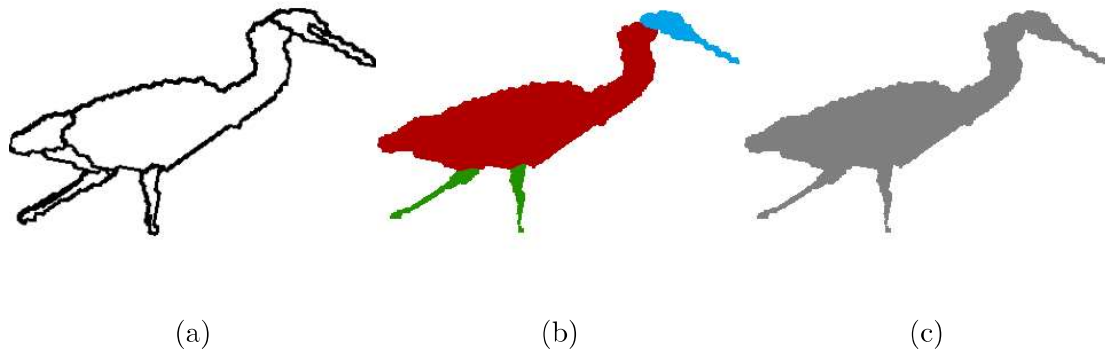


Figure 4.5: (a) Sketch example of the SketchParse dataset. (b) Annotation of the sketch where each colour corresponds to a body part. [SDBM17]. Blue denotes the head, red the torso and green the limbs. (c) Annotation transformed to a single label for training the detection step.

x are passed to the end. There the results of the layers are added element-wise with the original values of x . This allows the network to differentiate between the desired output and the current intermediate representation, meaning that the gradients can be updated more effectively during learning.

We want to detect hand-drawn characters in the input. This means that we need to define two classes, one for a hand-drawn character and one for the background, where nothing is present. For the backbone, first stage, of the Mask R-CNN we use the 50-layer version of ResNet, which consists of 48 convolutional layers (combined in residual building blocks), one max pool layer and one average pool layer. As an output of the network we define the standard classification and bounding box regression layers defined by pytorch [PGM⁺19].

For the bounding box predictor and the classification, we use the standard classification and bounding box regression layers defined by pytorch [PGM⁺19] for Fast R-CNN with two classes and 1024 features. For the mask output, we use the standard definition of Mask R-CNN by pytorch [PGM⁺19], with two classes, which are the foreground and background, 256 hidden layers and 256 features.

We use the SketchParse dataset provided by Sarvabebhatla et al. [SDBM17] for training our detection network. The dataset provides sketches with annotations for each part of the sketch, such as legs, body and head. Figure 4.5(a) shows an example for a sketch of the dataset with the annotation of the parts shown in Figure 4.5(b). Since we do not need to differentiate the sketch parts at the detection step, we can transform the different labels into a single one (Figure 4.5(c)).

To create training data containing multiple drawings we distribute multiple sketches of the dataset onto a single input image. Each of the sketches is randomly rotated to increase the diversity of the training data, as shown in Figure 4.6. The same transformation is

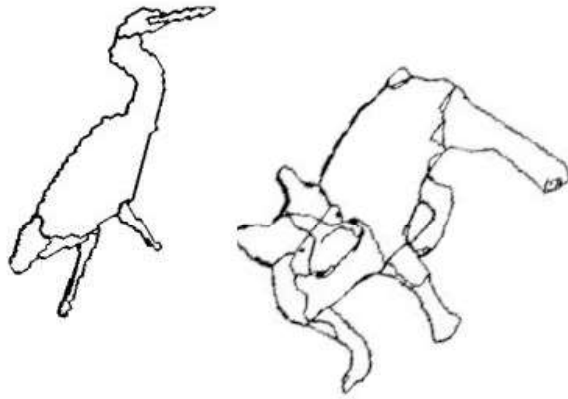


Figure 4.6: Input for the training step using multiple randomly rotated and placed sketches of the SketchParse dataset [SDBM17].

applied to the annotation, which allows the fast creation of a large amount of training data.

Based on the input (Figure 4.7(a)) for each detected character a bounding box is calculated (Figure 4.7(b)). Additionally, for each bounding box, an exact mask (Figure 4.7(c)) is created which can later be re-used during skeletonisation. For the following pipeline steps each of the detected hand-drawn characters is processed individually.

4.3 Semantic Segmentation

In the second step of the pipeline, as shown in Figure 4.1(c), we extract the semantic information of the hand-drawn character. The input of this step is the region of interest of the image of the previously detected hand-drawn character. The goal of this step is to get information about each part of the sketch and therefore which function the body part takes over, e.g. limbs are used for walking. The output of this step is an image, where each pixel contains a label for a semantic area, is one of the following:

- Body (i.e., torso)
- Head
- Limbs (i.e., legs and arms)
- Wing
- Tail

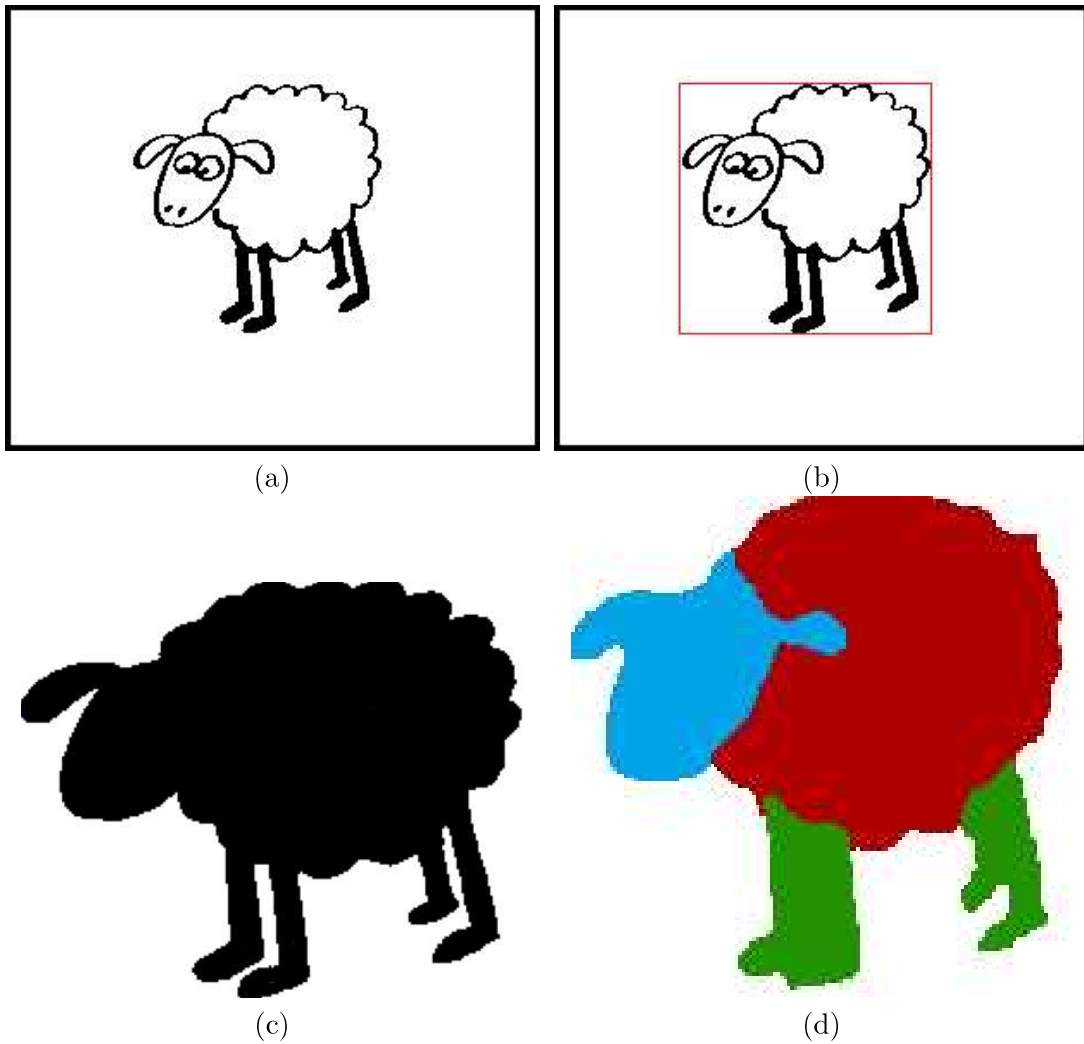


Figure 4.7: (a) Input containing a hand-drawn character. (b) The predicted bounding box (in red) of a detected character. (b) Predicted mask of a detected character inside the bounding box. (c) Semantic segmentation of a character with limbs (green), head (blue) and torso (red).

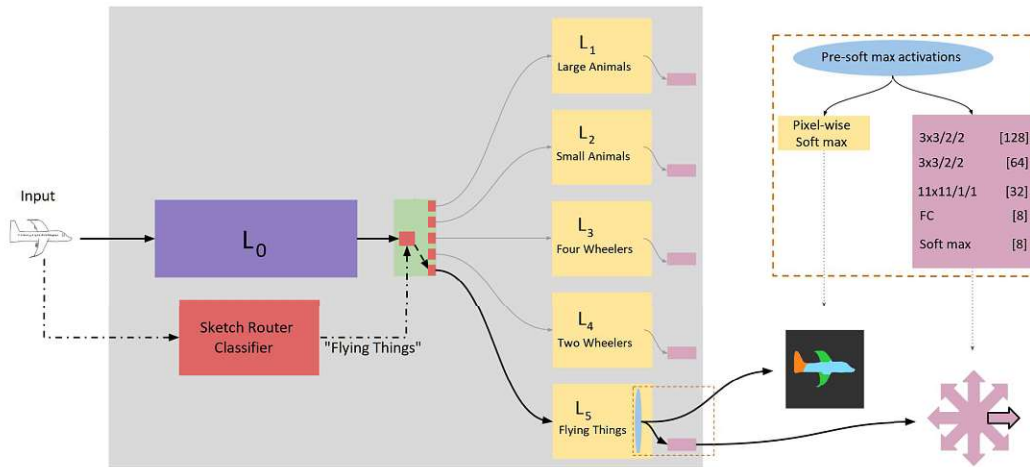


Figure 4.8: SketchParse architecture proposed by Sarvabebhatla et al. [SDBM17].

For this task, we can partially apply the SketchParse algorithm proposed by Sarvabebhatla et al. [SDBM17] using the novel architecture for a two-level fully convolutional network. Figure 4.8 shows the basic architecture of the multi-level network.

As an input we can use the image extracted from the detected bounding box, as shown in Figure 4.7(b). The common first layer, described as L_0 in Figure 4.8, extracts category-agnostic low-level information. The second layer consists of multiple specialised networks, each of them trained for a specific object category (L_1, L_2, \dots, L_5 in Figure 4.8). To define which sketch should be passed to which specialised network, they introduce a router layer which is implemented as a K -way classifier, shown in red in Figure 4.8. The K -way classifier is a CNN trained on actual sketches, that labels the input image to be forwarded to the specialised networks that themselves label the sub-parts of the hand-drawn characters.

Additionally to the pixel-wise labels, the 2D pose of the sketch is estimated. The pose describes in which direction the sketch is facing. The network differentiates between eight different directions, as shown in Figure 4.8.

For this thesis, due to the limited categories of characters, we only employ two specialised networks for characters that can semantically differentiate between the previously mentioned parts, namely body, head, limbs, wing, and tail. Figure 4.7(c) shows the segmentation of the detected character.

4.4 Character Skeletonisation

The next step in the pipeline, as shown in Figure 4.1(d), is to create the skeleton based on the mask computed in the detection step (Figure 4.7(c)). We use the computed skeleton to animate the hand-drawn character. The output of this step is the skeleton, of the

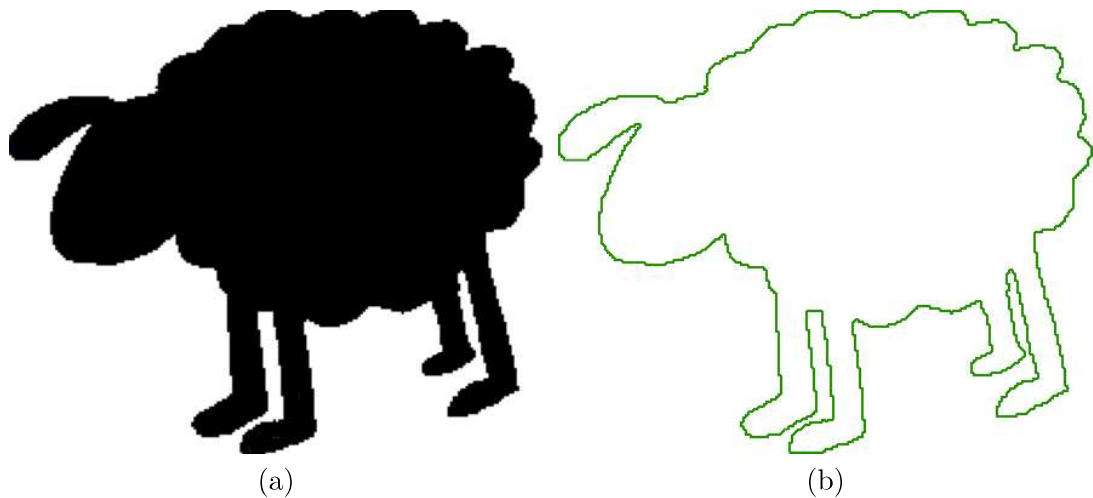


Figure 4.9: (a) Binarized input, black defines 1 and white defines 0. (b) Contour (green) of the character computed using the approach by Suzuki and Abe [SA85].

character, where we assign each joint and end-effector the label of the underlying pixel computed in the previous step.

First, we compute an initial skeleton using the Voronoi-based approach, as described in Chapter 3.4. The input requires the contour of the object.

To compute the contour we follow Suzuki and Abe [SA85] who propose a widely known algorithm for contour detection. First, we transform the mask of the object computed in the detection step (4.7(b)) into a boolean matrix, where one, shown black in Figure 4.9(a) corresponds with the foreground that should be skeletonised and zero (white) representing the background and provide it as an input to the contour detection algorithm.

As an output, we retrieve the contour, marked green in Figure 4.9(b), for the given character. This contour is exactly one pixel wide, at each point, and we use it as an input to the Voronoi-based skeletonisation algorithm. As an output of the Voronoi-based skeletonisation we get an image, where 1 indicates the pixel belongs to the skeleton and 0 indicates the pixel is not part of the skeleton.

It is possible that the algorithm returns non-thin parts, meaning parts of the skeleton are more than one pixel wide. For later steps of the pipeline, specifically the identification step, we need to ensure a one-pixel-wide skeleton. To ensure this property, we can follow the approach described by Zhang et al. [ZS84] which is described in Algorithm 4.1.

The basic idea of the algorithm is to iterate over the whole image and analyze the one-by-one neighbourhood of each point $P_1(i, j)$, indexed by row i and column j in the image, as shown in Figure 4.10.

The thinning algorithm is split into two sub-iterations with the first one deleting a pixel $P_1 = (i, j)$ if all of the following conditions apply

Algorithm 4.1: Thinning algorithm by Zhang et al. [ZS84]

Data: I : Skeleton represented as a matrix, where 1 indicates the pixel belonging to the skeleton and 0 otherwise

```

1 Set all elements of  $M$  to 0;      /* Matrix of the same size as  $I$  */
2 do
3   Set  $C = 0$ ; /* Counts if any pixels fulfill the conditions
   */
4   foreach Pixel  $P_1(i, j)$  in  $I$  do
5      $M(i, j) = 1$  if all conditions  $(C_a), (C_b), (C_c), (C_d)$  apply
6     if  $M(i, j) == 1$  then
7        $C+ = 1$ 
8     end
9   end
10   $I = I - M$ 
11  if  $C == 0$  then
12    stop
13  end
14  Set  $C = 0$  foreach Pixel  $P_1(i, j)$  in  $I$  do
15     $M(i, j) = 1$  if all conditions  $C_a, C_b, C_c, C_d$  apply
16    if  $M(i, j) == 1$  then
17       $C+ = 1$ 
18    end
19  end
20   $I = I - M$ 
21 while  $C != 0$ ;

```

P ₉ I(i-1, j-1)	P ₂ I(i-1, j)	P ₃ I(i-1, j+1)
P ₈ I(i, j-1)	P ₁ I(i, j)	P ₄ I(i, j+1)
P ₇ I(i+1, j-1)	P ₆ I(i+1, j)	P ₅ I(i+1, j+1)

Figure 4.10: Neighbourhood of the pixel P_1 [ZS84].

(C_a) $2 \leq B(P_0) \leq 6$, ensuring the endpoints of a skeleton are preserved

(C_b) $A(P_1) = 1$, preserving points that lie between the endpoints of a skeleton line (defined as the pixels of the skeleton connecting two points)

(C_c) $P_2 * P_4 * P_6 = 0$, for preserving eastern endpoints

(C_d) $P_4 * P_6 * P_8 = 0$, for preserving southern endpoints

with $A(P_1)$ being the number of 01 patterns in the ordered set of neighbouring points, e.g. given the ordered neighbours $P_2 = 0, P_3 = 1, \dots, P_9 = 0$, e.g. the pattern 01000001 results in $A(P_1) = 2$. $B(P_1) = \sum_{i=2}^9 P_i$. If all of the above conditions apply, the pixel is marked as 1 in the matrix M and the counter C is increased by one. In the second sub-iteration the conditions (C_c) and (C_d) are substituted by the following conditions

($C_{c'}$) $P_2 * P_4 * P_8 = 0$, for preserving northern endpoints

($C_{d'}$) $P_2 * P_6 * P_8 = 0$, for preserving western endpoints

with (C_a) and (C_b) staying the same. After each iteration, the matrix M is subtracted from I removing pixels that do not belong to the optimal one-pixel wide skeleton. If the counter C is zero after an iteration the optimal skeleton has been found and the algorithm stops, returning I . Figure 4.11(a) shows the skeleton, marked with white pixels, obtained after applying the thinning algorithm overlaid onto the shape, represented in black.

The output I contains the skeleton of the character. While the thresholding during the Voronoi-based skeletonisation, as described in Chapter 3.4 removing spurious skeleton lines, Figure 4.11 shows that there are still many undesired end-effectors and joints in the skeleton, e.g. end-effectors inside the body of the semantic type *body* (red in Figure 4.11(a)), can lead to errors in further pipeline steps. Therefore, our aim is to remove as many artefacts before moving to the next pipeline step.

For further processing we represented the skeleton as a graph G_{Skel} . At each endpoint of a skeleton line, we can insert a vertex, which represents an end-effector. At pixels where multiple skeleton lines meet, we can also insert a vertex representing a joint. Figure 4.11(a) shows the inserted joints and end-effectors coloured depending on the semantic labels extracted in the previous step (Chapter 4.3). Vertices that lie next to each other on a skeleton line are connected with an edge in G_{Skel} .

We propose two heuristics, H_1 and H_2 , to remove those undesired artefacts. For H_1 our idea is to remove all end-effectors of the semantic type *body* and *head* that we do not want to consider in future pipeline steps, e.g. the head of a character is one component and does not contain multiple parts. H_1 contains the following condition for removing a vertex u :

(C_1) Vertex u has exactly 1 incident edge

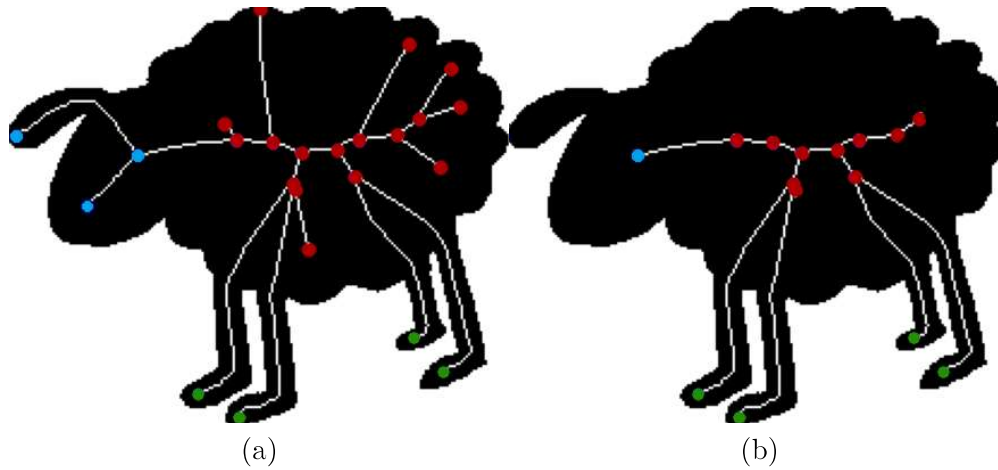


Figure 4.11: (a) Skeletonised character with bones (white) and joints and end-effectors (red = type *body*, blue = type *head*, green = type *limb*) (b) Heuristically simplified skeleton by removing end-effectors of type *head* and *body* if they are incident to another node of the same type.

(C_2) Vertex v , the neighbour of u , has the same type $type(v) = type(u)$

(C_3) $type(v) = head$ or $type(v) = body$

The function $type_v$ returns the semantic type, e.g. *body* or *head*, for the given vertex v . This means the conditions (C_1), (C_2) and (C_3) we remove all end-effectors (vertices with exactly one neighbour), that are of the type *body* or *head* and the only neighbouring vertex is of the same type. Figure 4.11(b) shows the skeleton after applying H_1 removing all of the undesired end-effectors. Specifically condition (C_2) ensures that we do not remove the only vertex of the given type, i.e., *head*.

Another issue arises due to the nature of Voronoi skeletonization generating branches very close together, as the multiple vertices of type *body* shown in Figure 4.12(b). Vertices so close together increase the complexity of the skeleton while providing little to no benefit for further computations.

To avoid this problem, we propose H_2 to merge any vertices of the same type, e.g. two vertices of type *body*, if they are below a given threshold ϵ , which we define as 5 pixels. The distance between two vertices $P_1(i, j)$ and $P_2(k, l)$, with i and k being the row of the pixel in the image and j and l being the column, is calculated using the Euclidean distance, shown in Equation 4.1.

$$d(P_1(i, j), P_2(k, l)) = \sqrt{(i - k)^2 + (j - l)^2} \quad (4.1)$$

If two vertices P_1 and P_2 have a distance $d(P_1(i, j), P_2(k, l))$ lower than the threshold ϵ we merge the two vertices to one vertex P_n with the position given by Equation 4.2.

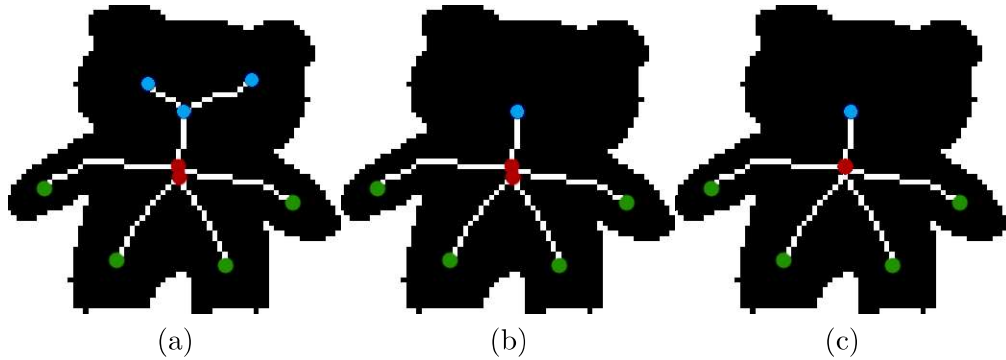


Figure 4.12: (a) Skeletonised character with bones (white) and joints and end-effectors (red = *body*, blue = *head*, green = *limb*) (b) Skeleton after applying H_1 . (c) Skeleton after applying H_2 .

We connect all vertices that were previously connected with an edge to P_1 or P_2 with an edge to P_n .

$$P_n = \left(\frac{i+k}{2}, \frac{j+l}{2} \right) \quad (4.2)$$

Our experiments showed different weighting of the new position did not lead to any significant differences but added another parameter and complexity, therefore a simple average position is preferred.

Figure 4.12(a) shows another example of a character skeleton retrieved by Voronoi skeletonisation with the vertex types assigned by semantic segmentation. In the first step, we apply H_1 which results in Figure 4.12(b) and in the second step we apply H_2 to merge the two nodes of the type *body* that are close together in the centre of the sketch.

The output of this pipeline step is the graph G_{Skel} that is generated from the Voronoi skeletonisation and simplified by applying the heuristics H_1 and H_2 . Additionally, for each edge e we save the skeleton line, defined as the pixels connecting the two vertices in I , that is generated as it will be used in the next pipeline step.

4.5 Character Identification

In the next step of the pipeline, as shown in Figure 4.1(e), we identify the reference skeleton that maps best to the given skeleton. The input of this pipeline step is the graph G_{Skel} computed in the previous step. The output of this step is the best-matching reference skeleton that represents a base movement type.

For this thesis, we limit ourselves to three different reference skeletons, each with a unique movement style:

- Humanoid, characters moving similar to a human, i.e., walking on two legs and upright
- Quadruped, characters moving typically on four legs
- Flying, characters moving typically using wings and legs

The number of reference skeletons is not limited in general but can be expanded by defining more reference skeletons. If the new reference skeletons contain new types of parts, e.g. claws, it is also necessary to expand the training data for semantic segmentation in section 4.3 to make it possible to extract information about the new parts.

For each of the reference skeletons we can define animations which can later be applied to the hand-drawn character. To match the skeleton generated in the previous step with the reference skeletons we can follow the approach proposed by Cheong et al.[CGK⁺09].

Let us assume that we define our prototype skeletons as graphs G_{hum} , G_{quad} and G_{fly} for humanoid, quadruped and flying reference skeletons. Figure 4.13(a) shows our definition of a humanoid skeleton, marking end-effectors in green for representing *limbs*, joints in red representing the *body* and end-effector in blue representing the *head*. Respectively Figure 4.13(b) and Figure 4.13(c) show the reference skeleton for quadruped and flying reference skeletons. Further, for each of the characters we define a simple shape which is used in later pipeline steps.

Therefore, the task for this step is defined as finding the closest match between G_{skel} and the three reference skeletons G_{hum} , G_{quad} and G_{fly} . We propose to use a graph-based approach for this matching problem.

For computing which of the prototype skeletons matches closest to the unknown skeleton, we base our approach on the findings of Cheong et al.[CGK⁺09] who provide a novel algorithm for measuring the similarity of geometric graphs, called geometric graph distance (ggd).

Their algorithm is based on the graph edit distance that has been discussed in section 3.2. The graphs we retrieved from the skeletons have an inherent geometric meaning, due to the position of the vertices in the image plane. This makes the geometric graph distance a fitting approach to match the given graphs.

While the operations in the graph edit distance are allowed to be in an arbitrary order, for the geometric graph distance the following operations are available and need to be done in the following order:

1. Edge deletions
2. Vertex deletions
3. Vertex translations

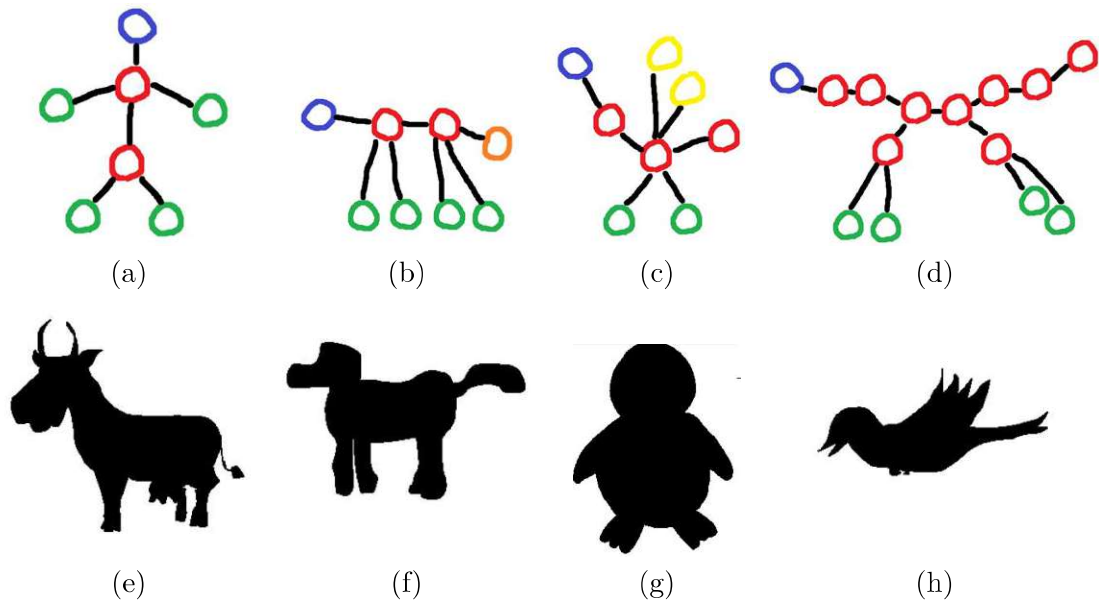


Figure 4.13: Red denotes *body* vertices, green for *limb* vertices, yellow for *wing* vertices, orange for *tail* vertices and blue for *head* vertices. (a) Reference skeleton for a humanoid character. (b) Reference skeleton for a quadruped character. (c) Reference skeleton for a flying character. (d) Skeleton obtained from the previous step. (e-h) Show diverse example shapes that are used to generate the skeletons.

4. Vertex insertions

5. Edge insertions

The ordering of the operations is necessary as they ensure that the geometric graph distance is symmetric. This means that vertex translations have to appear in the middle. Only isolated vertices can be deleted, meaning that edges have to be deleted before vertices. Further, Cheong et al. [CGK⁺09] argue that deletions after insertions are never useful.

The geometric graph distance is defined as follows: let us assume two graphs $G_A = (V_A, E_A)$ and $G_B = (V_B, E_B)$. We aim to find a subset of vertices $V^* \subseteq V_A$, as vertices can be deleted or inserted, and a set of operations $\sigma : V^* \rightarrow V_B$, e.g. deleting a vertex $v \in V_A$, to transform the graph G_A to G_B . We define two constants for costs, namely K_e and K_v which are associated with operations on edges and vertices respectively. The costs for operations in σ are defined as follows,

- (a) the cost for any edge $(u, v) \in E_A$ such that not both $u, v \in V^*$ or $\sigma(u)\sigma(v) \notin E_B$ is $K_e|uv|$, where K_e is a constant cost and $|uv|$ is the length of the edge

- (b) the cost for any edge $(u, v) \in E_B$ such that not both $u, v \in \sigma V^*$ or $\sigma^{-1}(u)\sigma^{-1}(v) \notin E_A$ is $K_e|uv|$
- (c) the cost for each edge $(u, v) \in E_A$ with both $u, v \in V^*$ and $\sigma(u)\sigma(v) \in E_B$ is $K_e||uv| - |\sigma(u)\sigma(v)||$
- (d) the cost for each $u \in V^*$ is $K_v|u\sigma(u)|$, where K_v is a constant cost and $|u\sigma(u)|$ is the Euclidean distance between the node position u and $\sigma(u)$.

The geometric graph distance is defined as the minimum of this cost over all choices of V^* and σ . Cheong et al. [CGK⁺09] proved that the computation of the geometric graph distance is NP-hard and can only be solved by formulating the problem as an integer linear programming (ILP) problem and using a mixed integer linear programming (MIP) solver.

The following shows how the ILP problem is defined. First, we define a binary variable, referred to as V_{uv} representing pairs of vertices $u \in V_A$ and $v \in V_B$. V_{uv} is set to one if $u \in V^*$ and $\sigma(u) \in V_B$, otherwise zero. All pairs of edges $e \in E_A$ and $e' \in E_B$ are represented by the binary Variable $E_{ee'}$, that is set to one if both $u, v \in V^*$ and $e' = (\sigma(u), \sigma(v))$.

The objective function is defined as in Equation 4.3 using constants K_e and K_v . $|uv|$ describes the Euclidean distance between two nodes. $|e|$ describes the length of the edge $e = (u, v)$ which is determined by the Euclidean distance of the two vertices u and v .

$$\begin{aligned}
ggd &= K_v \cdot \sum_{u,v} (|uv| \cdot V_{uv}) \\
&+ K_e \cdot \sum_{e \in E_A} (|e|) \\
&+ K_e \cdot \sum_{e' \in E_B} (|e'|) \\
&- K_e \cdot \sum_{e,e'} (|e| + |e'| - ||e| - |e'||) \cdot E_{ee'}
\end{aligned} \tag{4.3}$$

This means that the first term of the objective function $K_v \cdot \sum_{u,v} (|uv| \cdot V_{uv})$ calculates the sum of distances between two matched nodes u and v . The second and third terms calculate the cost of deleting all edges in G_A and inserting all edges of G_B . Deleting and inserting all edges can be avoided by moving an edge $e \in G_A$ to G_B which is represented in the fourth term.

To avoid matching a vertex $u \in V_A$ to multiple vertices in V_B and vice versa, we need to introduce the following constraints shown in Equation 4.4 and Equation 4.5.

$$\forall u \in V_A, \sum_v^{V_B} V_{uv} \leq 1 \tag{4.4}$$

$$\forall u \in V_B, \sum_v^{V_A} V_{uv} \leq 1 \quad (4.5)$$

Additionally, we need another constraint to avoid an edge $e \in E_A$ being mapped to multiple edges $e' \in E_B$. Equation 4.6 shows the constraint on the edges.

$$\begin{aligned} \forall e(u, v) \in E_A \\ \forall e'(u', v') \in E_B \\ E_{ee'} \leq 0.5 \cdot (V_{uu'} + V_{vv'} + V_{uv'} + V_{vu'}) \end{aligned} \quad (4.6)$$

We extend the original definitions presented by Cheong et al. [CGK⁺09] to include the labels of the nodes in the geometric graph distance. This allows us to utilise the semantic information extracted in section 4.3 to match the skeletons more robustly.

For this, we modify the objective function of Equation 4.3 by adding another cost K_l assuming $lbl_dist(u, v)$ returns the distance between the labels of the nodes $u \in V_A$ and $v \in V_B$. This distance function is configurable and if the distance between any combination of labels is one, it is analogue simply counting the label differences of matched nodes. This results in Equation 4.7. Our experiments show that the label information allows for more robust matching results.

$$\begin{aligned} ggd = & K_v \cdot \sum_{u,v} (|uv| \cdot V_{uv}) \\ & + K_e \cdot \sum_{e \in E_A} (|e|) \\ & + K_e \cdot \sum_{e' \in E_B} (|e'|) \\ & - K_e \cdot \sum_{e,e'} (|e| + |e'| - ||e| - |e'||) \cdot E_{ee'} \quad + K_l \cdot \sum_{u,v} (lbl_dist(u, v) \cdot V_{uv}) \end{aligned} \quad (4.7)$$

The described algorithm allows us to compute the ggd's for all pairs of skeletons, specifically, $ggd(G_{skel}, G_{quad})$, $ggd(G_{skel}, G_{hum})$ and $ggd(G_{skel}, G_{fly})$. As an output for this step we use the reference skeleton with the lowest ggd. In our pipeline, the graphs that need to be matched are relatively simple and contain below ten nodes. This means that the ILP problem can be solved very fast. Cheong et al. [CGK⁺09] found for large graphs the algorithm is not applicable anymore and suggest using a heuristic matching algorithm, such as the Landmark Distance [CGK⁺09].

While our experiments show that the matching to the closest reference skeleton works very robustly, the mapping of the end-effectors is not reliable using the given method, but absolutely necessary for computing the animation. This issue is resolved in the last pipeline step.

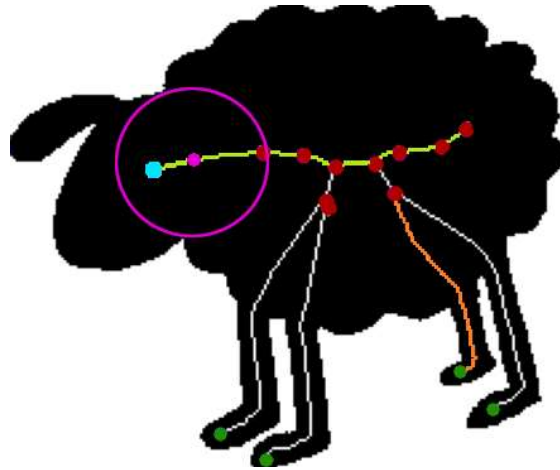


Figure 4.14: Examples for two skeleton paths marked in green and orange on the skeleton denoted in white. Pink indicates a sample point on the skeleton path with its associated maximal disc centred on the sample point while still contained inside the shape.

4.6 Rigging and Animation

The last step of the pipeline, as shown in Figure 4.1(f), consists of multiple operations. The input of this step requires the original image as an input, the skeleton of the character and the reference skeleton to create the animation for the character.

In the previous step, we identified the closest matching reference skeleton to the skeleton of the character G_{skel} . While for very small graphs brute-forcing a good mapping is possible, we provide a more robust way. Especially for complex skeletons, meaning a larger number of end-effectors makes brute-forcing no longer feasible. For robust mapping of the end-effector nodes, which is necessary for animating the character, we follow the algorithm Bai and Latecki [BL08] for matching skeleton paths. The approach is very robust, as it does not rely on inner vertices, e.g. the joints of our skeleton, but only on the end-effectors. A skeleton path is defined as a sequence of pixels of the skeleton connecting two vertices of the graph, as shown in Figure 4.14. We denote the two skeletons that we want to map as G_A and G_B for this section.

The basic idea of the algorithm is, that a mapping of two skeletons is robustly computed by comparing the skeleton paths. Further, they take into account the thickness of the shape (black in Figure 4.14).

The thickness of the shape at each point of the skeleton is given by a maximal disc which is still contained by the shape. To approximate the maximum disc for each sample point on the skeleton as marked in pink in Figure 4.14. The maximum disc is approximated using the distance transform, such as proposed by Rosenfeld and Pfaltz [RP66]. As a first step, we calculate the distance transform for $G_A = (V_A, E_A)$ and $G_B = (V_B, E_B)$ based on their respective shapes, e.g. Figure 4.14 shows the shape of G_{skel} . Figure 4.15(a)

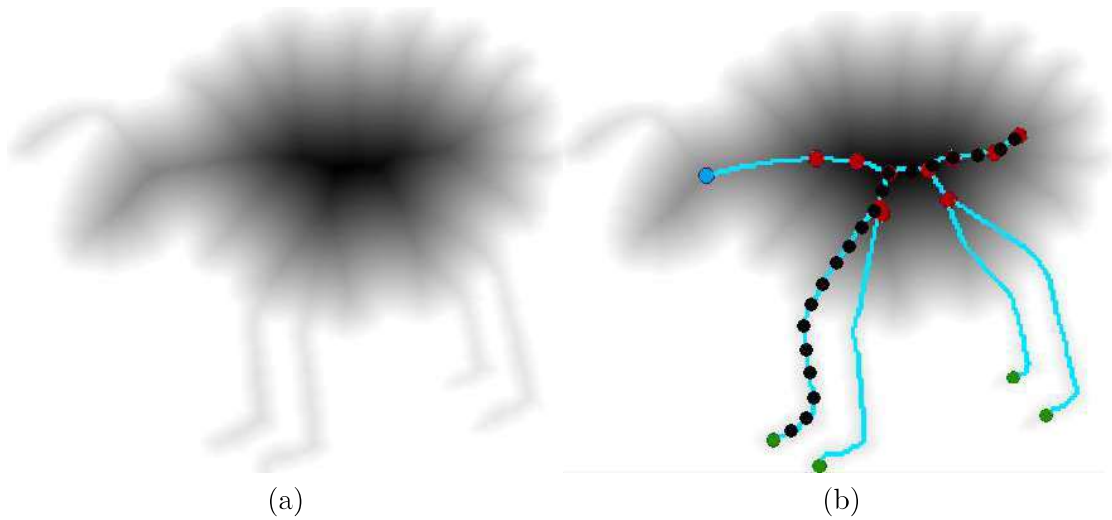


Figure 4.15: (a) Distance transform computed for each pixel on the shape of the character. Lower intensity indicates a higher distance to the boundary of the shape. (b) Uniformly distributed points (black) on the shortest skeleton path connecting two end-effectors. Turquoise lines represent the skeleton obtained by the previous step.

shows the distance transform of each pixel of the shape of G_A , where lower intensity indicates a higher distance to the shape boundary.

For each pair of end-effectors, e.g. $u, v \in V_A$, we compute the shortest path using the shortest path algorithm proposed by Edsger Dijkstra [Dij22]. Along each skeleton path, we create M uniformly distributed sample points along the skeleton paths that connect the two end-effectors as shown in Figure 4.15(b). While some edges are processed multiple times, the distance transform calculations done are computationally cheap and are therefore negligible. Further, multiple processes of the same edge are necessary to ensure a uniform distribution of M sample points. For each of the sample points, we sample the value of the distance transformation which we denote as the function $DT(p)$ on the pixel position p of the sample point. To make the approach invariant to scale the sampled value is normalized using Equation 4.8, where N represents the number of all pixels and p_i the pixel position.

$$R_p = \frac{DT(p)}{N \sum_{i=1}^N DT(p_i)} \quad (4.8)$$

To find the best matching skeleton paths, denoted as $p(u, v)$ with $u, v \in V_A$ and $p(u', v')$ and $u', v' \in V_B$, which is necessary to compute the mapping of the end-effectors we calculate the path distance. The path distance $pd(p(u, v), p(u', v'))$ is given by Equation 4.9. M denotes the number of sample points, r_i the radius of the maximal disc at the i -th sample point of $p(u, v)$ whereas r'_i denotes the i -th sample point of $p(u', v')$. l and l' define the length of the skeleton path $p(u, v)$ and $p(u', v')$ respectively. The length of

the skeleton path is exactly computed by summing the distances between two sequential pixels of the skeleton path. Our experiments showed that the approximation by counting the pixels did not lead to different mapping results. Lastly, the configurable constant α is used to control the influence of the length of the skeleton paths, which after experiments we set to 0.1.

$$pd(p(u, v), p(u', v')) = \sum_{i=0}^M \frac{(r_i - r'_i)^2}{r_i + r'_i} + \alpha \frac{(l - l')^2}{l + l'} \quad (4.9)$$

To find the optimal mapping of the end-effectors of both graphs, we compute the path distance for each combination of vertices $u \in V_A$ and $u' \in V_B$. First, we obtain an ordered list of end-effectors for both graphs, resulting in a sequence u_0, u_1, \dots, u_n and u'_0, u'_1, \dots, u'_k . We order the end-effectors by their angle with respect to the centroid of all end-effectors. Algorithm 4.2 shows that the centre can be calculated by summing up all vertices and dividing the result by the number of end-effectors. Then a common sorting algorithm can be used, such as Quicksort proposed by Charles Hoare [Hoa62], to order the end-effectors based on Algorithm 4.3.

Algorithm 4.2: Ordering nodes clockwise based on their angle.

Data:

EF : end-effectors of the skeleton

S : ordered set of end-effectors

```

1 set  $p_{centre}$  to  $(0, 0)$ 
2 foreach vertex  $u$  in  $EF$  do
3   |  $p_{centre} += u$ 
4 end
5  $p_{centre} = p_{centre} / size(EF)$ 
6  $S = sort(p_{centre}, EF, compareVertices)$ ; /* Sorting algorithm that
   orders elements based on the output of function shown in
   Algorithm 4.3 */
7 return  $S$ ; /* Ordered end-effectors  $u_0, u_1, \dots, u_n$  */
```

The angle of an end-effector, relative to the given centre, be calculated using Equation 4.10 with u representing the position. p_{center} represents the centroid, as calculated earlier. If two end-effectors have the same angle, we order them using their respective distance from the centre using the Euclidean distance shown in Equation 4.1. As a result, we retrieve the ordered list of end-effectors.

$$angle(u, p_{centre}) = arctan2(u(1) - p_{centre}(1), u(0) - p_{centre}(0)) \quad (4.10)$$

The ordered lists allow us to compare all combinations of path distances for all end-effectors, defined as $d(u_0, u'_0)$ for two end-effectors u_0 and u'_0 . The combinations of path distances can be written as the matrix in Equation 4.11.

Algorithm 4.3: Compares two end-effectors based on their angle towards a given centre.

Input:

p_{centre} : Centre relative to all vertices

u : Vertex

v : Vertex

Output: true if u should be sorted before v , false otherwise

```

1 Function compareVertices ( $p_{centre}, u, v$ ):
2    $angle_u = angle(u, p_{centre})$ 
3    $angle_v = angle(v, p_{centre})$ 
4   if  $angle_u < angle_v$  then
5     | return true
6   end
7    $dist_u = euclidean(u, p_{centre})$ 
8    $dist_v = euclidean(v, p_{centre})$ 
9   if  $angle_u == angle_v$  and  $dist_u < dist_v$  then
10    | return true
11  end
12  return false
13 End Function

```

$$d(u_0, u'_0) = \begin{bmatrix} pd(p(u_0, u_1), p(u'_0, u'_1)) & \dots & pd(p(u_0, u_1), p(u'_0, u'_k)) \\ pd(p(u_0, u_2), p(u'_0, u'_1)) & \dots & pd(p(u_0, u_2), p(u'_0, u'_k)) \\ \vdots & \vdots & \vdots \\ pd(p(u_0, u_n), p(u'_0, u'_1)) & \dots & pd(p(u_0, u_n), p(u'_0, u'_k)) \end{bmatrix} \quad (4.11)$$

Bai and Latecki [BL08] suggest that the cost of matching two end-effectors u_0 and u'_0 is computed using the optimal subsequence bijection presented by Latecki et al. [LWKTM07]. We define the cost as $c(u, u') = OSB(d(u, u'))$, where OSB is a function that returns the optimal subsequence bijection distance given the matrix of path distances $d(u, u')$.

According to Bai and Latecki [BL08] the optimal subsequence distance can be computed by using a shortest path algorithm on a directed graph. The vertices of the directed graph are all index pairs $(i, j) \in \{1, 2, \dots, n\} \times \{1, 2, \dots, k\}$. The weight w of an edge in the graph is defined by Equation 4.12.

$$w((i, j), (k, l)) = \begin{cases} pd(u_i, u'_j) & i + 1 = k \text{ and } j + 1 \leq l \\ (k - i - 1) \cdot k & i + 1 < k \text{ and } j + 1 \leq l \\ \infty & \text{otherwise} \end{cases} \quad (4.12)$$

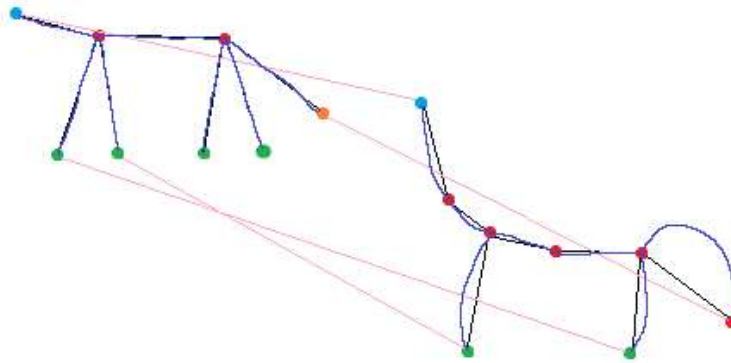


Figure 4.16: Mapping (pink) of the end-effectors of the reference skeleton (left) and the detected hand-drawn character (right). Lines in blue show the actual skeleton lines, while black shows the bones (simplified) connecting the joints and end-effectors.

The constant k defines a constant cost of skipping the matching of a given vertex, i.e., if the two graphs have a different number of vertices. Equation 4.13 shows how the cost is determined. This means that for every vertex u_i the closest vertex u'_j of which we take the mean plus one standard deviation, std , of the distances of the closest vertices.

$$k = \text{mean}_i(\min_j(pd(u_i, u'_j))) + \text{std}_i(\min_j(pd(u_i, u'_j))) \quad (4.13)$$

Based on the results of the *OSB* for each combination of vertices, we get the resulting matrix $c(G_A, G_B)$. The optimal mapping can be found by using the Hungarian algorithm, which is a common formulation for globally optimal matching. The Hungarian algorithm requires a square matrix, therefore we add the last row in Equation 4.14.

$$c(G_A, G_B) = \begin{matrix} & u'_0 & u'_1 & u'_2 & u'_3 \\ \begin{matrix} u_0 \\ u_1 \\ u_2 \end{matrix} & \begin{pmatrix} 0.5 & 0.7 & 0.9 & 1.2 \\ 0.3 & 0.4 & 3.5 & 4.0 \\ 0.1 & 4.0 & 2.1 & 17.0 \end{pmatrix} \\ & \infty & \infty & \infty & \infty \end{matrix} \quad (4.14)$$

Figure 4.16 shows the mapping of end-effectors between a detected hand-drawn character (right) and its, in the previous step matched, reference skeleton of a quadruped character. It shows the capability of matching even if the skeletons have a different number of end-effectors.

Before we can animate the hand-drawn character, we need to create a mesh based on the image to use for the skeleton-based animation. For creating a mesh there are multiple options we can choose from. One option is to re-use the Voronoi diagram that we calculated in section 4.4. Using the Voronoi regions inside the contour we can create a mesh. Figure 4.18(b) shows the drawback of this approach. The mesh faces created this

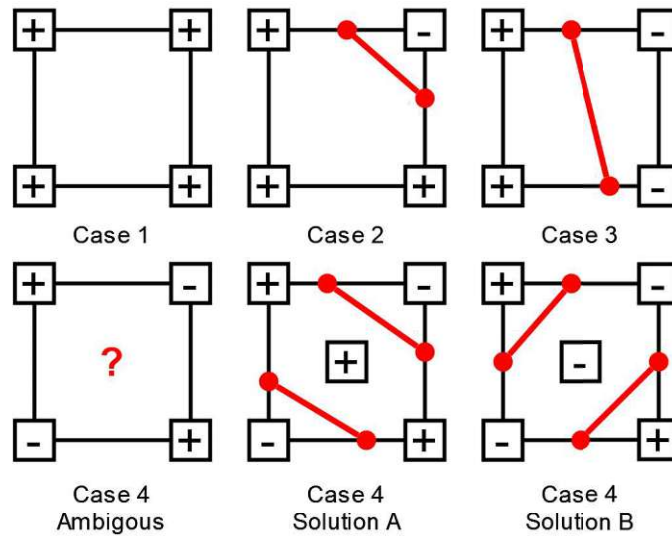
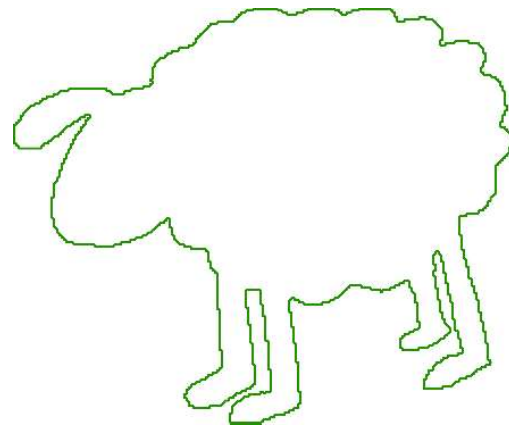


Figure 4.17: Four cases of the marching squares algorithm. + and - indicate whether the corner of the voxel is inside or outside the shape, while the squares represent the voxels. Red-marked are new edges and vertices added in each case.

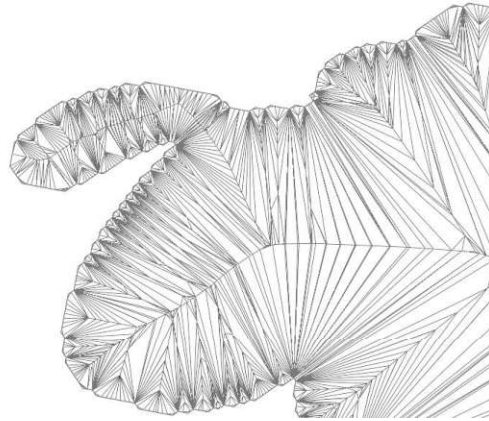
way are long and thin, which results in the mesh tearing when animated if the vertices of the face have very different weights compared to their neighbours, due to their distance from each other.

We can achieve a better result by following a voxel-based approach to produce a uniform mesh. The basic idea is that we divide the 2D space into a grid of square voxels. We can then use the marching squares as presented by Carsten Maple algorithm [Map03] to create the mesh. The basic idea is to iterate over each voxel and identify the value of the four corners, which are either inside (black) or outside (white) defined by the shape of the character shown in Figure 4.14. Figure 4.17 shows the four possible cases after taking symmetries into account. The edge marked in red is additionally inserted to the mesh. In the first case, all corners are inside and the vertices, represented by the corners of the voxel, are added to the mesh, but no additional edge is inserted. For cases two and three one edge is inserted based on the configuration shown in Figure 4.17. Case four is not trivial as the solution is ambiguous if only the four corner points are considered. Therefore, we check whether the value inside the voxel is inside or outside and add two edges and vertices accordingly. The position of the additional vertices is approximated by linear interpolation. Figure 4.18(c) shows the generated mesh using the voxel-based approach.

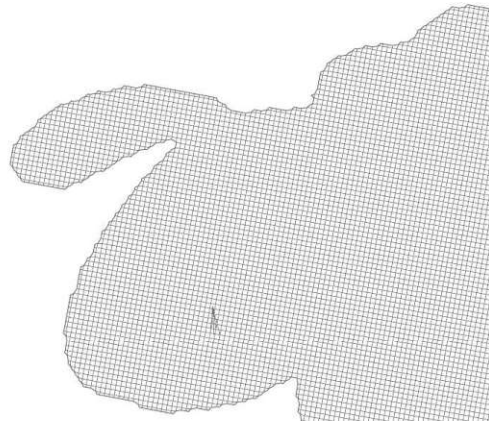
As the next step, we assign a weight for each mesh vertex, which we generated, that indicates the influence of the bones of the animated reference skeleton. This is necessary so that the mesh deforms according to the skeleton-based animation. For this step, we can rely on the semantic segmentation computed in section 4.3 and shown in Figure 4.7(c). The end-effector of the skeleton extracted from the character is mapped to an end-effector



(a)



(b)



(c)

Figure 4.18: (a) Contour of the shape determining the area that should be triangulated. (b) Triangulation using Voronoi regions shows undesired thing triangles. (c) Quad mesh using a voxel-based approach.

of the reference skeleton, which corresponds to the bones of the animated reference skeleton.

The weight assignment can be done by using a flood-fill approach, such as presented by James Foley [Fol96]. Let us assume that the mesh of the detected character represents a Graph $G = (V, E)$, where mesh vertices represent vertices $v \in V$ and mesh edges represent edges $e \in E$.

Algorithm 4.4: Weight assignment to vertices of the generated mesh.

Data:
 σ : weight to be assigned
 $G = (V, E)$: graph representing the mesh
 I : matrix containing the semantic label for each pixel
 EF : end-effectors of the detected characters skeleton

```
1 set  $S$  to empty set ; /* Set containing nodes to visit */
2 foreach end-effector  $ef \in EF$  do
3   set  $vis$  to empty set ; /* Set containing visited nodes */
4   pick closest vertex  $u \in V$  to  $ef$ 
5   put  $u$  in  $S$ 
6   set  $l$  to label at position of  $u$  in  $I$ 
7   foreach vertex  $v \in S$  do
8     assign  $\sigma$  to  $v$ 
9     add  $v$  to  $vis$ 
10    foreach neighbouring vertices  $w \in V$  to  $v$  do
11      if  $w \in V$ ; /* ( $C_1$ ) */
12      then
13        | continue
14      end
15      if  $label(w, I) \neq l$ ; /* ( $C_2$ ) */
16      then
17        | continue
18      end
19      if  $dist(w, ef) \leq dist(w, ef') \forall ef' \in EF$ ; /* ( $C_3$ ) */
20      then
21        | add  $w$  to  $S$ 
22      end
23      else
24        | add  $w$  to  $vis$ 
25      end
26    end
27  end
28 end
```

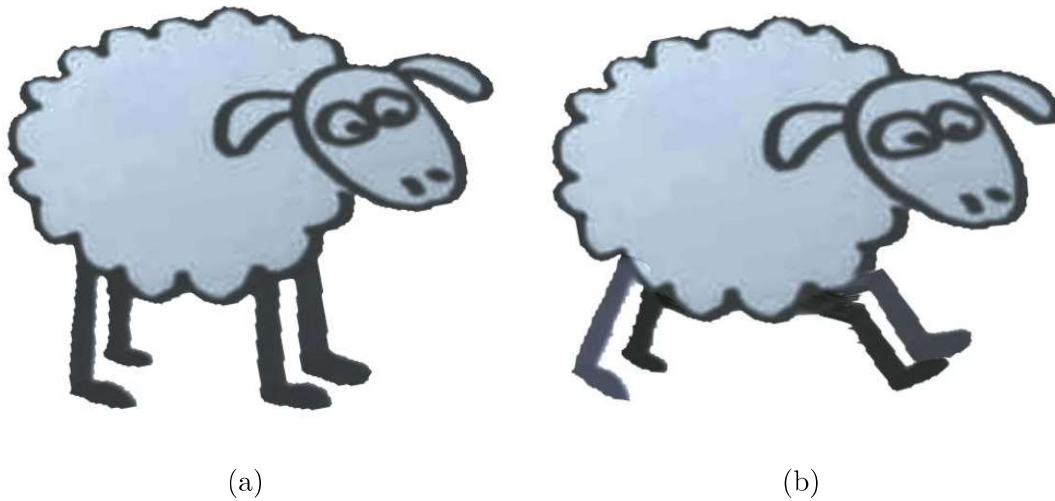


Figure 4.19: (a) Final 2.5D model generated from a detected hand-drawn character. (b) The animated character using retargeted animation of the reference skeleton.

Following the Algorithm 4.4 for each end-effector we pick the closest node $u \in V$ and assign the weight σ for the corresponding bone to the mesh vertex. We determine the weight between $[0.0, 1.0]$ depending on the ratio of lengths of the detected characters' bone (connecting the end-effector) and the length of the reference skeletons bone.

We continue in a breadth-first search manner visiting all neighbouring nodes. We need to introduce conditions to avoid assigning weights to vertices that should not be affected by the movement of a bone, i.e., vertices inside the torso should not be deformed when moving the leg. Therefore, we continue assigning weights for the vertices to this end-effector as long as we can find a neighbouring vertex $w \in V$ that matches the following conditions:

- (C_1) Neighbouring vertex w has not been visited yet
- (C_2) Neighbouring vertex w has the same label as the closest end-effector
- (C_3) Neighbouring vertex w is closest to the current end-effector of the same label

Condition (C_1) ensures that we only assign weights to vertices that have not been visited yet. Condition (C_2) ensures that we only assign weights of end-effectors with the same semantic label as the vertex. E.g. we avoid assigning weights of a limb to vertices of the torso. The last condition (C_3) makes sure that vertices are only assigned to one bone of the same type, e.g. we should avoid assigning vertices weights for multiple legs. Figure 4.7(d) shows that the semantic segmentation will sometimes result in, i.e., labels for the legs being connected through small parts of the torso.

As the last step, the texture coordinates of the vertices can be simply calculated by normalizing the vertex position using the size of the input image. The texture is given by the input image so that the final model contains all hand-drawn details. After this step, the hand-drawn character uses the animations defined for the reference skeleton. Figure 4.19 shows the final output of the pipeline, with the mesh of the character on the right and the deformed mesh on the left, where the deformation is given by the pose of the animated reference skeleton.

Prototype Implementation

The prototype for evaluating the pipeline is implemented in Python version 3.10 on a Laptop with an Apple M2 Chip (8 Cores @ 3.5GHz, 16MB L3 Cache) and 8GB RAM. Machine learning codes are realised with Pytorch 1.0.2 [PGM⁺19], whereas graph-related data structures and standard algorithms such as searching for shortest paths are using NetworkX 2.6.3 [HSSC08]. The average runtime for each step of the pipeline for the tested hand-drawn characters of the dataset is shown in Table 5.1. The size of the input images is 640×480 . The bounding box, which determines the image size for segmentation, of the detected characters, is on average 284×260 . After skeletonisation, the number of vertices is between seven and 15 with an average of eleven vertices and an average of twelve edges. The heuristics remove an average of three vertices. We tested on a set of 50 hand-drawn characters that will be discussed in Chapter 6.

For the detection of the hand-drawn characters, described in Section 4.2, we use the network as defined in the section, relying on the definition of the Mask R-CNN and Fast R-CNN by pytorch [PGM⁺19]. For training the character detection, we use stochastic gradient descent and the following parameters, as suggested by pytorch [PGM⁺19],

- Learning Rate: 0.005
- Momentum: 0.9
- Weight Decay: 0.0005
- Gamma (multiplicative factor of learning rate decay): 0.1

Table 5.1: Shows the runtime of each pipeline step in milliseconds (ms).

Detection	Segmentation	Skeletonisation	Identification	Rigging
1532ms	1020ms	420ms	911ms	1510ms

- Step size for learning rate decay: every third epoch

We use 2000 epochs for training the network and use the SketchParse dataset provided by Sarvadevabhatla [SDBM17] as described in section 4.2 to generate training data.

For the semantic segmentation as described in section 4.3, for this thesis, we reuse the public code provided by Sarvadevabhatla et al. [SDBM17]. Since the codes were using an old version of Python we adjusted the code, i.e., formatting and functions no longer available, to be compatible with Python version 3.10. We provide the changed source files in our GitHub repository [Kor23]. We made no changes to the general network architecture or how the network is trained. The network is trained on images with the size 321×321 , which is the size of images provided in the SketchParse dataset [SDBM17]. We found during experiments using that when applying grey erosion, provided by scipy [VGO⁺20] improved the segmentation when the input is based on a photograph of a real drawing. The basic idea is that this makes the input more similar to the data that was used for training, containing sharp edges. Further, we rely again on the definition of ResNet-50 by pytorch [PGM⁺19] for computing the feature map as suggested by Sarvadevabhatla et al. [SDBM17] and use stochastic gradient descent for training and use the default parameters of the public code,

- Learning Rate: 0.005
- Momentum: 0.9
- Gamma (learning rate lr factor depending, on the epoch e and maximum number of epochs e_{max}): $g = lr * (1.0 - \frac{e}{e_{max}})^{0.9}$
- Maximum number of epochs: 20000

which allowed us to replicate the results presented by Sarvadevabhatla et al. [SDBM17].

For the skeletonisation we use the algorithm described in section 4.4, we use a threshold of 30, for pruning the Voronoi-based Skeleton using the chord residual, for images of the size 321×321 pixels and adjust it linearly for smaller images. We do this to avoid pruning the skeleton too aggressively for smaller inputs.

When matching the skeleton of the hand-drawn character to the reference skeletons, as described in section 4.5 we determine the optimal parameters for computing the ggd by testing the success rate on 50 hand-drawn characters. The best results are achieved when defining $K_v = 1.0$, $K_e = 1.0$ and $K_l = 5.0$ when matching with G_{hum} and G_{fly} . When comparing to G_{quad} define $K_n = 0.38$ while keeping the other parameters the same. While the original ggd proposed by Cheong et al. [CGK⁺09] uses absolute parameters for the matching, we found during experiments that different-sized input images can influence the success rate negatively. We found by normalizing the coordinates of the vertices to an interval $[0, 1]$ leads to the best results. For the label distance lbl_dist we

found the best result by sweeping the parameter. We define it as the absolute differences of the values assigned to the labels as follows,

- Head is assigned 1
- Body is assigned 2
- Tail is assigned 4
- Limb is assigned 7
- Wing is assigned 10

This means that i.e., the cost of changing the label from *head* to *tail* is associated with a cost of 3. Finally for solving the ILP, as defined in section 4.5 we use the python implementation by cvxopt [ADV20] that uses the GNU Linear Programming Kit [Oki12]. As ϵ , we define 0.00005 for stopping the optimisation and a maximum number of 10000 iterations.

After identifying the best matching reference skeleton based on the results of *ggd* and mapping the end-effectors as described in section 4.5, we compute the mesh and assign the influence of bones on each mesh vertex as described in Chapter 4. The vertex position and texture coordinates, as well as the influence information of each bone on each vertex is saved in the *obj* file format and an auxiliary *txt* file.

Finally, for generating the animated hand-drawn character, we use Blender [Com18] to demonstrate the results. In Blender, we define the reference skeletons and animate them. Through the scripting interface of Blender, we load the computed *obj* file that contains the computed vertices and texture coordinates as well as the *txt* file that contains the weight information for each of the vertices for each bone. After assigning the influence of bones on each vertex, we demonstrate the capabilities of the pipeline in Blenders embedded rendering engine.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Results and Discussion

In this chapter, the results of the pipeline are presented and the details are discussed. Afterwards, the limitations of the current pipeline are discussed. We publish the code on Github [Kor23] which includes examples of animated hand-drawn characters.

6.1 Experimental Results

The prototype implementing the methodology is done in Python, as described in Chapter 5. For each reference skeleton, humanoid, quadruped and flying, an animation is created using Blender [Com18]. The following figures show selected frames of the animated hand-drawn character output by the pipeline. The following figures show selected frames from left to right and top to bottom of the animated character. The characters are illuminated with a light source in the rendering engine of Blender.

For quadruped characters, the reference skeleton is animated to move the legs left and right while moving the tail in the third dimension (parallel to the character) forward and backward. The head is animated to move upwards and downwards. For humanoid characters, the reference skeleton is animated to move the limbs (arms and legs) forward and backward while the head moves left and right.

The Figures 6.1, 6.2, 6.3 and 6.4 show correctly identified quadruped characters. One issue encountered with the giraffe character is that the legs, due to self-occlusion in the drawing, are not detected separately during the skeletonisation leading to only two limbs being assigned, but the result is still reasonable with respect to the input.

Figure 6.5 and 6.6 show two correctly identified humanoid characters. The motion applied is walking forward. Figure 6.7 shows a fish character that has been assigned, quadruped motion since there is no reference type for fishes. The lower fins are mapped to the legs of the quadruped reference skeleton, while the upper part of the head is mapped to the head of the reference skeleton. The fin in the back is mapped to the tail.

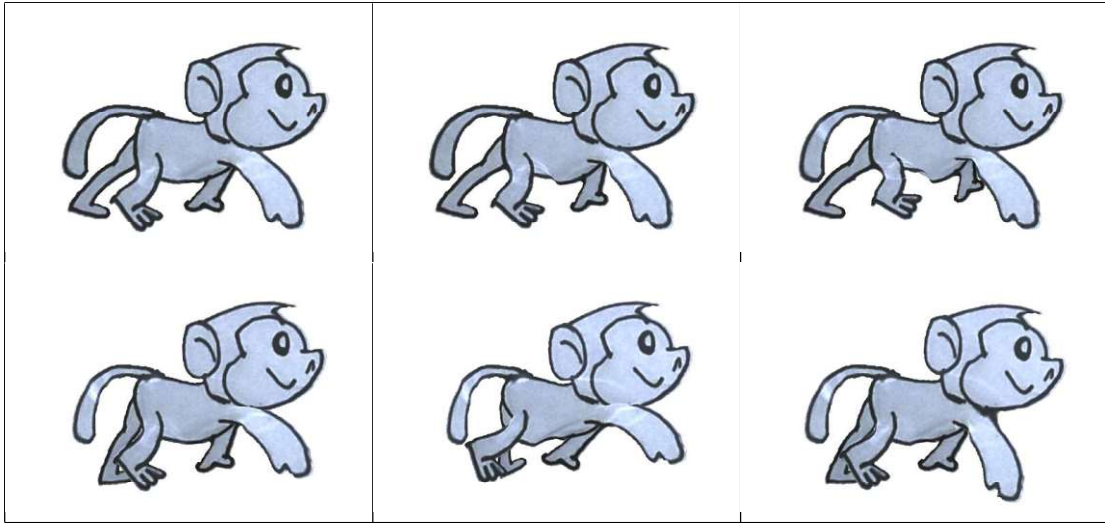


Figure 6.1: Six selected consecutive (from left to right and top to bottom) frames of the example movement applied to the hand-drawn monkey character.

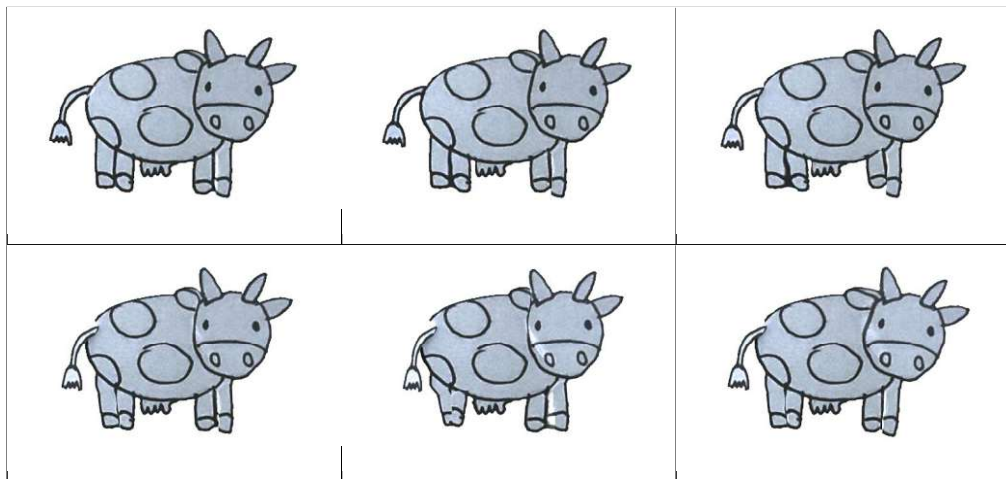


Figure 6.2: Six selected consecutive (from left to right and top to bottom) frames of the example movement applied to the hand-drawn cow character.

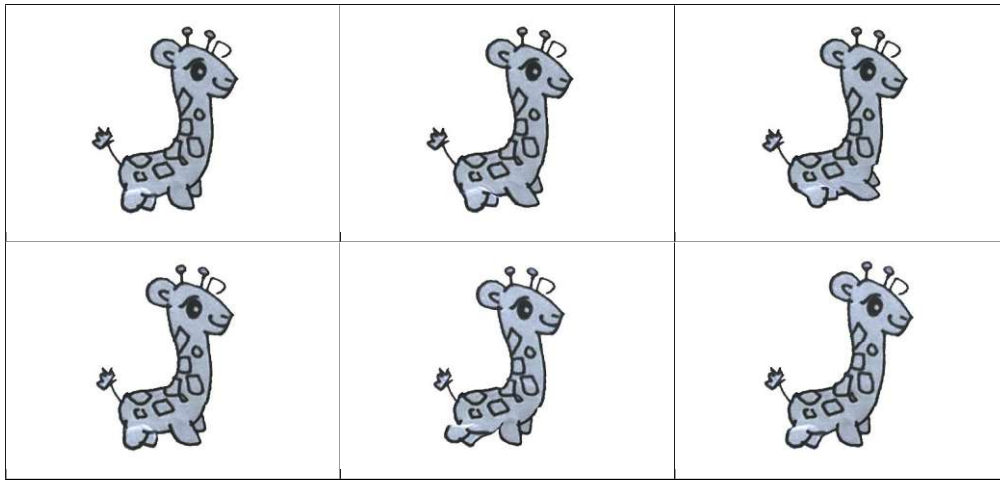


Figure 6.3: Six selected consecutive (from left to right and top to bottom) frames of the example movement applied to the hand-drawn giraffe character.

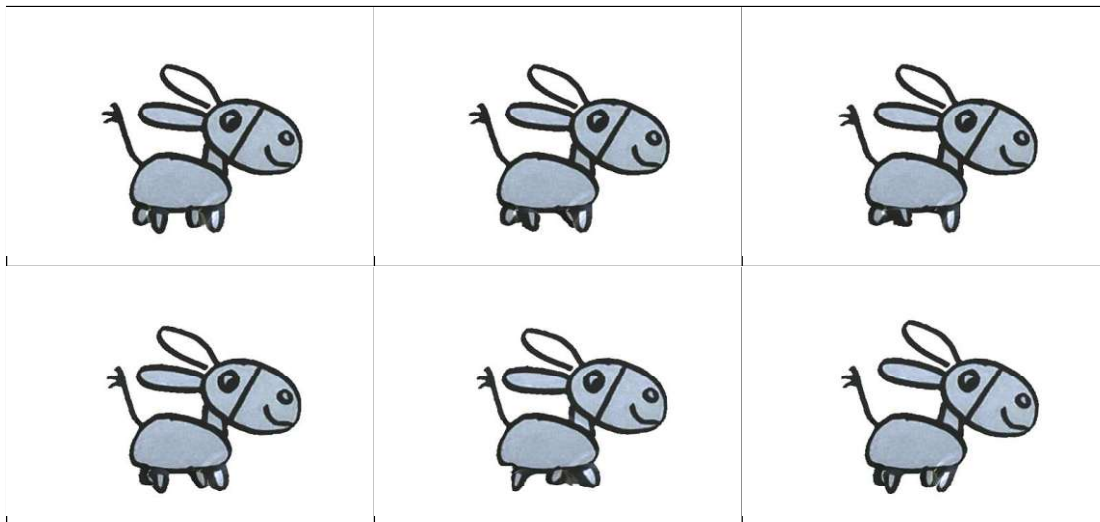


Figure 6.4: Six selected consecutive (from left to right and top to bottom) frames of the example movement applied to the hand-drawn donkey character.

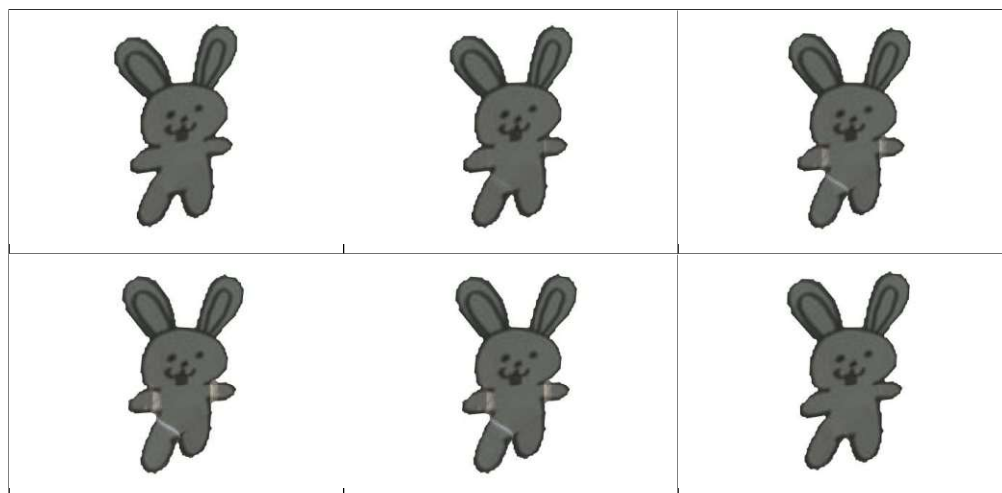


Figure 6.5: Six selected consecutive (from left to right and top to bottom) frames of the example movement applied to the hand-drawn bunny character.

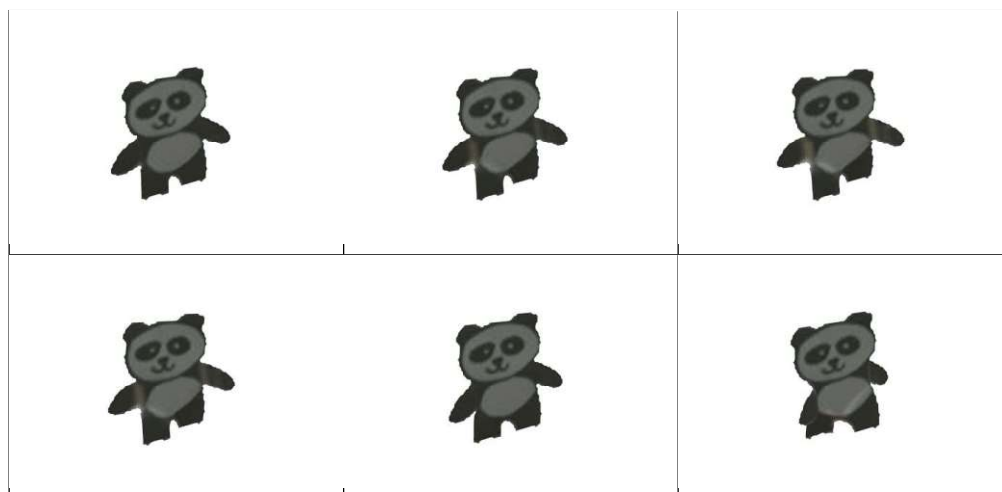


Figure 6.6: Six selected consecutive (from left to right and top to bottom) frames of the example movement applied to the hand-drawn panda character.

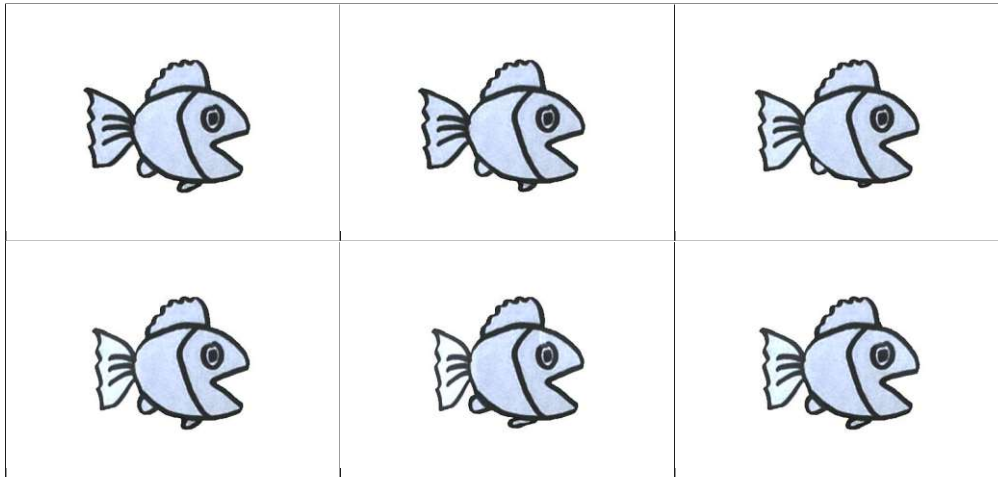


Figure 6.7: 6 selected frames of the example movement applied to the hand-drawn fish character.

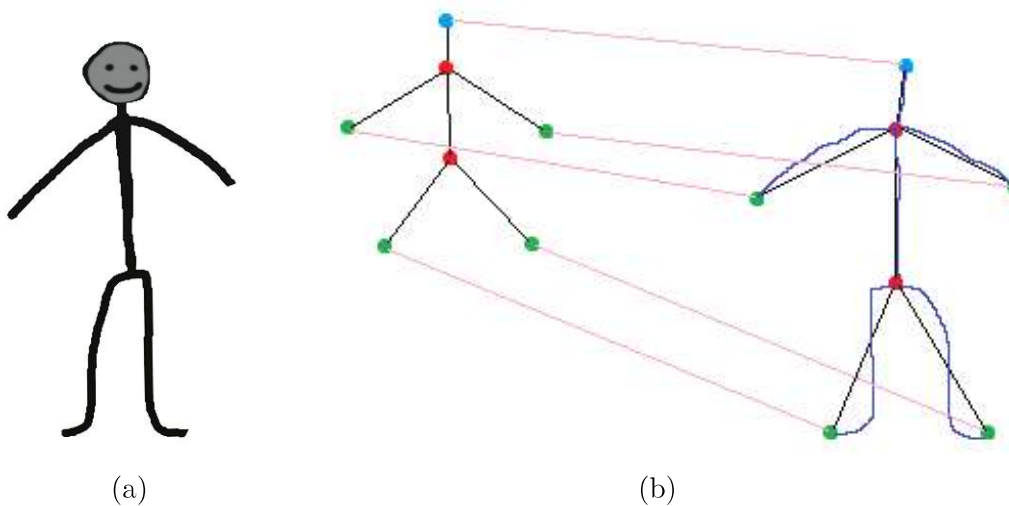


Figure 6.8: (a) Shows a simple stickfigure character. (b) After detection and identification as a humanoid character, the end-effectors are mapped correctly to the prototype skeleton.

Compared to previous examples, also less articulate hand-drawn characters, such as the simple stickfigure as seen in Figure 6.8(a) is correctly identified as a humanoid character and correctly mapped end-effectors (right) to the reference skeleton (left) as seen in Figure 6.8(b).

6.2 Evaluation and Discussion

We evaluate the proposed approach with a set of 50 hand-drawn characters, where the input is mixed between digital hand-drawn characters and analogue input. The hand-drawn characters vary in quality between clear articulate drawings and drawings that have fewer details. Tables 6.1, 6.2, and 6.3 gives a selection of the set, where the first column names the ground truth type, humanoid, quadruped and flying, of the character and the second column shows the input image. Further columns indicate the success or failure of each pipeline step.

On the dataset of 50 hand-drawn characters, all characters can be detected. For the segmentation using the algorithm proposed by Sarvadevabhatla [SDBM17] as described in Chapter 4 we can get the same performance as described in their paper.

The identification of the best matching reference skeleton for the given hand-drawn character after skeletonisation we reach an accuracy of 85.00%. Compared to state-of-the-art classification approaches such as Sketchnet proposed by Zhang et al. [ZLZ⁺16] that classifies sketches into multiple categories, which reaches an accuracy of around 80% on 20 classes. Another comparison can be made to the evaluation of the original ggd implementation that reached an accuracy of 94.5% for matching Chinese characters [CGK⁺09] of 6 different fonts, where each character is represented by a graph. To the best of our knowledge the task of matching skeletons, represented by labelled graphs, has not yet been explored in other research.

Another observation we made is that flying characters are always correctly identified in our sample data. Humanoid and quadruped characters seem to be less robust, which is due to the similarity of the two reference graphs, especially for cases where no tail is present or not correctly identified by the semantic segmentation network. One failing example is shown in row three of Table 6.1 where an actually well-articulate drawn dog is identified as a humanoid moving character with a probability of 88.57% compared to a probability of being quadruped with 88.31%. Such failures can often be traced back to slightly incorrect semantic segmentation. As in the case of the dog, the tail is detected as a limb, which if manually corrected would be correctly identified as a dog. Although we observe that the difference in probabilities of assigning a character as humanoid or quadruped are small throughout the dataset with around 4%, which we explain due to the similarity of the reference skeletons.

6.3 Limitations of the Pipeline

One limitations of our presented pipeline is the discrepancy between the generated mesh and the skeleton. For example given Figure 6.9 the skeleton is created with all four limbs and correctly mapped to the quadruped reference skeleton. But the mesh is created on the shape of the drawing and is not influenced by the skeleton. In this case, it leads to the vertices of the two limbs, e.g. in the front, to be connected, which leads to stretching effects as highlighted in Figure 6.9. There is also the other way around where 2 limbs

Table 6.1: Table shows selected results of the proposed approach. A ✓ indicates the success of the given pipeline step, while a ✗ indicates that this pipeline step failed. Inputs marked with † indicate that they are taken from the SketchParse dataset of Sarvadevabhatla et al [SDBM17].

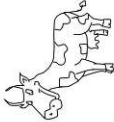

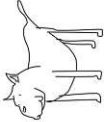




Ground Truth	Input	Detection	Segmentation	Skeletonisation	Identification	Rigging
Quadruped†		✓	✓	✓	✓	✓
Quadruped†		✓	✓	✓	✓	✓
Quadruped†		✓	✓	✓	✗	✓
Quadruped†		✓	✓	✓	✓	✓
Flying†		✓	✓	✓	✓	✓
Flying†		✓	✓	✓	✗	✓
Undefined		✓	✓	✓	✗	✓

Table 6.2: Table shows selected results of the proposed approach. A ✓ indicates the success of the given pipeline step, while a ✗ indicates that this pipeline step failed. Inputs marked with † indicate that they are taken from the SketchParse dataset of Sarvadevabhatha et al [SDBM17].















Ground Truth	Input	Detection	Segmentation	Skeletonisation	Identification	Rigging
Quadruped†		✓	✓	✓	✓	✓
Quadruped†		✓	✓	✓	✓	✓
Flying		✓	✓	✓	✓	✗
Flying		✓	✓	✓	✓	✓
Humanoid		✓	✓	✓	✗	✓
Quadruped		✓	✓	✓	✓	✓
Quadruped		✓	✓	✓	✓	✓

Table 6-3: Table shows selected results of the proposed approach. A ✓ indicates the success of the given pipeline step, while a ✗ indicates that this pipeline step failed.

Ground Truth	Input	Detection	Segmentation	Skeletonisation	Identification	Rigging
Quadruped		✓	✓	✓	✓	✓
Humanoid		✓	✓	✓	✗	✓
Humanoid		✓	✓	✓	✓	✓
Humanoid		✓	✓	✓	✓	✓
Humanoid		✓	✓	✓	✓	✓
Quadruped		✓	✓	✓	✓	✓
Quadruped		✓	✓	✓	✓	✓

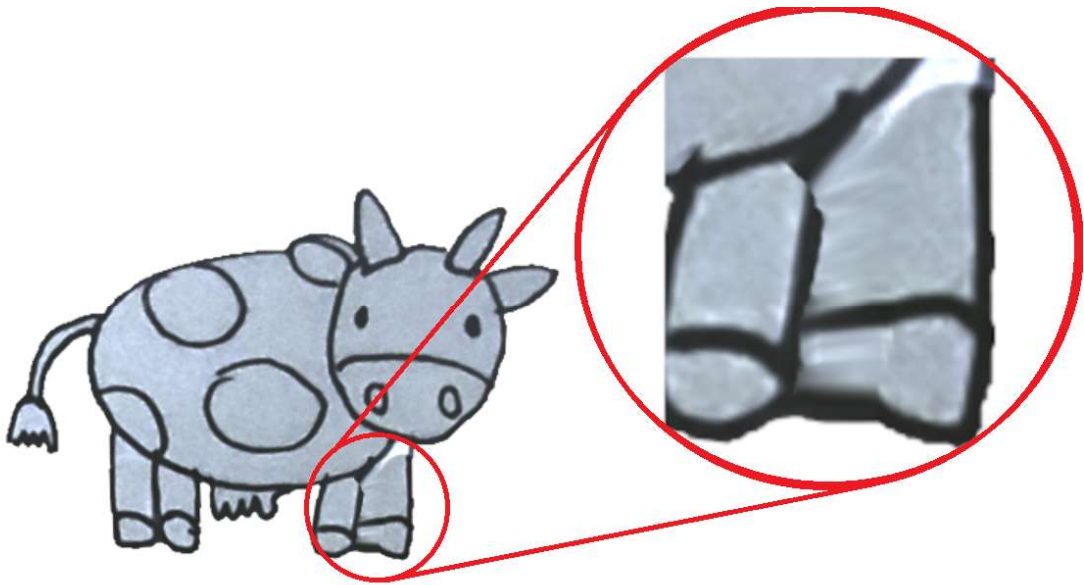


Figure 6.9: Shows the limitation if separate limbs are detected but the meshing does not consider them to be separate.

while drawn separately are not accurately split during skeletonisation but only one limb is identified, as can be seen in Figure 4.16.

Another limitation lies within the mapping of the end-effectors to the reference skeleton. We observed in some experiments that even though the characters are simple and similar in structure, the approach proposed by Bai and Latecki [BL08] is not very robust against symmetry as shown in Figure 6.10(a). While on symmetric motions this is no problem, it can lead to unexpected behaviour of the animated character when the movement is not symmetric.

For this thesis, we also restricted the input of possible hand-drawn characters to be on white (light) background with black (dark) foreground, as seen in Figure 4.2. This limitation can be removed when training the CNN used for detection on a wider range of sample input images.

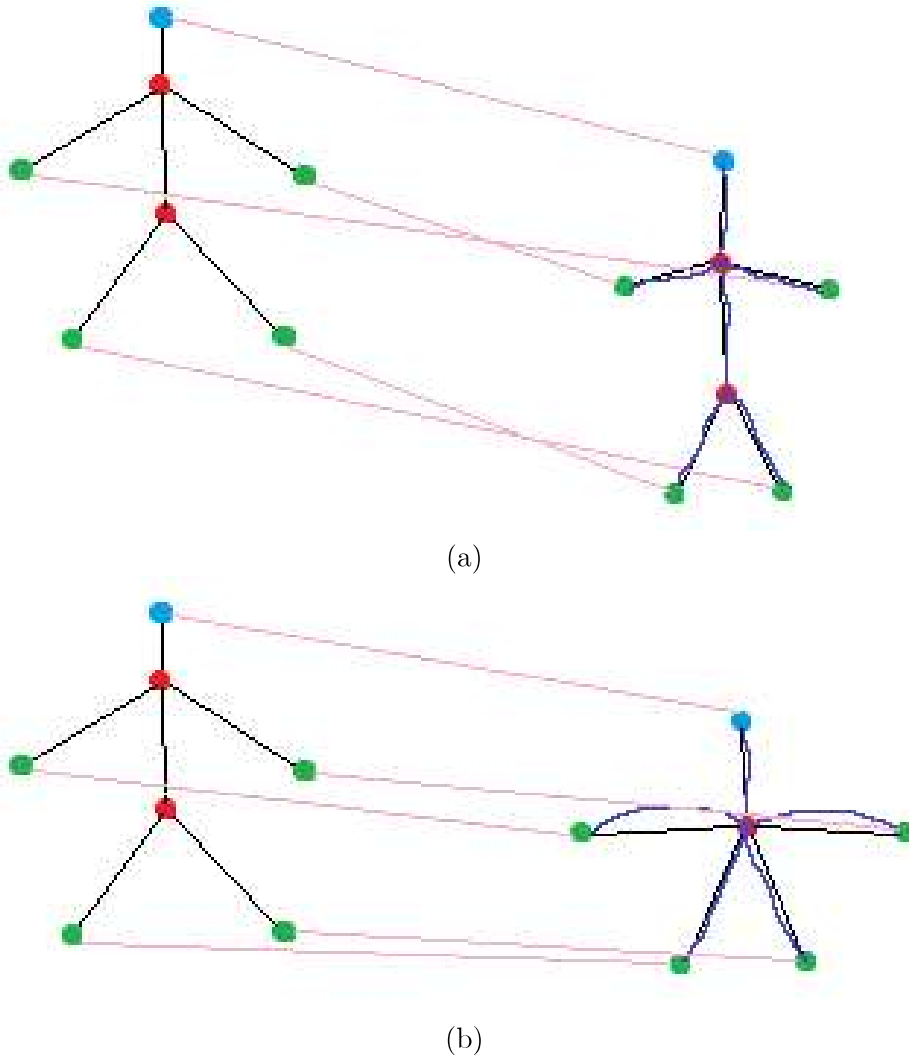


Figure 6.10: (a) Shows a character's skeleton end-effectors (right) wrongly assigned to the reference skeleton (left). (b) Shows the panda character's skeleton end-effectors (right) correctly assigned to the reference skeleton (left).



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Conclusion and Future Work

In this chapter, a short summary and conclusion of the thesis are provided. Last, we present ideas for future work.

7.1 Summary and Conclusion

We present a novel approach to animate 2D hand-drawn characters based on semantic properties extracted from the input. The pipeline consists of five steps, from detecting the hand-drawn character in an input image, to extracting the semantic information about the parts of the hand-drawn character, i.e., limbs, head, and body. After that, a skeleton is extracted and augmented with the semantic information extracted in the previous step. The skeleton is matched to a predefined reference skeleton that represents a certain type of skeleton, i.e., quadruped, humanoid or flying. In the last step the skeleton's end-effectors are matched to the prototype skeleton and the hand-drawn character is transformed into a model and rigged.

Our pipeline shows promising results for well-articulated hand-drawn characters. There is still room for improvement, especially with regard to mapping the end-effectors of the hand-drawn characters' skeleton and the reference skeleton. Another point for improvement is to find a more robust measure to differentiate between reference skeletons that are similar to each other, e.g. humanoid and quadruped skeleton, to avoid mismatches in case semantic segmentation does not give the correct labels for each character part. While the computed skeletons already show proof of the concept for generating semantic-aware animations for hand-drawn characters, the pipeline should be evaluated for higher complexity of skeletons and animations. The pipeline is also configurable making it easy to replace any of the steps with a better-performing algorithm to improve the performance of the overall pipeline.

7.2 Future Work

In future work, a user study should be conducted that evaluates whether the movement that is retargeted to the hand-drawn character is believable for the user. Further, it can be evaluated whether the hand-drawn character increases the engagement and retainment of information when using it during guided visualisations compared to pre-defined characters and no visual guide.

A point of improvement is to enable the possibility for more complex animations by improving the rigging in the last step. A potential idea is to super-sample the skeleton of the detected hand-drawn character and map joints based on the mapping information of the end-effectors.

Another consideration is to implement a pose estimation for the hand-drawn character. This could help to define more complex animations. Another consideration in future work can be that not only one reference skeleton is assigned to a detected hand-drawn character. In the case of fantastic animals, e.g. a gryphon is a combination of a lion and a hawk it can be interesting to interpolate the movement of a flying and quadruped character instead of specifying a specific reference skeleton.

For the first step, detecting the character, one improvement can be to be less restrictive on the input and allow more complex input, i.e., photographs of different angles of graffiti or drawings on cluttered backgrounds where it can be hard to differentiate between the foreground containing the character and background that should be ignored.

List of Figures

2.1	(a) A hand-drawn character is detected. (b) The detected character is brought to life as a textured 3D model in Augmented Reality. (c) A simple lively AR environment with multiple randomly animated characters [Jun].	6
2.2	(a) RGB Image as the input of the pipeline. (b) The detected hand-drawn character. (c) The segmentation mask separates the character from the background and the estimated joint key points (annotated in red). (d) Different animations applied to the hand-drawn character [SZL ⁺ 23].	7
2.3	Example of a sketch (top left) transformed in a 3D model and animated by dragging control points (red) with the mouse. [DSC ⁺ 20].	8
2.4	(a) The hand-drawn character consisting of seven body parts. (b) Computed major and minor axes of each part. (c) Oriented bounding boxes of each part. (d) Computed joint locations between the parts marked in blue. (e) Final computed skeleton with end-effectors inserted. [TBVDP04]	9
2.5	(a) Motion sketched for walking around two trees and leaping onto a platform. (b) Snapshots of the created animation for the character [TBVDP04]. . .	10
2.6	(a) Coloured dots annotate the position of the joints of the hand-drawn character by the user. (b) Segmentation of the body parts by the user. (c) Hand-drawn approximation of the bounding boxes by the user [JSMH12].	11
2.7	Different frames of the hand-drawn character augmented with cloth using the information of the virtual 3D proxy [JSMH12].	12
3.1	Basic structure of an Artificial Neural Network adapted from Keiron O’Shea and Ryan Nash [ON15].	15
3.2	(a) The two-dimensional input and the filter used in the convolution operation. (b) The filter is then moved step-by-step over the input, for example, the filter is overlaid with the input marked in blue for the first step and the calculated results are written to the output marked in red, to compute the results of the convolution operation. Adapted from Introduction to Convolutional Neural Networks [Ser23].	16
3.3	Max pooling operation on the given input produces the output on the right side, for a window with a stride of two and a size of two-by-two. The colours correspond to the window overlaid with the input and the output. . . .	18
3.4	Basic architecture of the Fast R-CNN architecture as proposed by Ross Girshick [Gir15].	18
		77

3.5	A simple undirected graph G represented by a node-link diagram. Each vertex is represented as a circle (node) and each edge is represented by a line (link) linking the two respective vertices.	20
3.6	The adjacency matrix representation of the graph G from Figure 3.5. . . .	20
3.7	Simple skeleton representing a human character. Circles describe joints, while the directed links between circles represent the bones connecting the joints.	22
3.8	(a) A hand and two bones of the associated skeleton. Red marks a mesh-vertex of the mesh representing the hand and its corresponding position relative to the bone. (b) Applying a rotation θ to the bone connecting the hand end-effector shows the change of the mesh and associated mesh-vertices. Adapted from what-when-how [Pub].	23
3.9	Skeleton-based deformation of a mesh based on two bones (red and green). The influence of the bones on each mesh-vertex is shown in the respective colour of the vertex [TP23].	24
3.10	A problem with two possible solutions for Inverse Kinematics [ALCS18]. The red circle marks the end-effector while blue dots mark the joint and the root. Joints, base and end-effectors are connected by bones.	25
3.11	Markerless motion capture of a human jumping [SGXT20]	26
3.12	Retargeting of a pose captured by markerless motion capture [SGXT20] .	27
3.13	(a) Example of a Voronoi diagram on a set of points marked in blue. (b) Voronoi diagram of selected points on the contour (blue) of a shape [LGK19].	28
3.14	Grey boxes show the contour of the shape and black lines show Voronoi edges. Each pixel of the contour is used as a site. For each point m the length of the chord length is calculated, meaning the distance of the contour connecting the closest two sites p_A and p_B [LGK19].	29
4.1	(a) Image containing hand-drawn character. (b) Detection step, where the character is separated from the background. (c) Semantic segmentation, where all parts of the sketch are identified. (d) Calculation of the skeleton. (e) Mapping of the skeleton to the best matching reference. (f) Triangulation of the sketch and rigging that results in the final animation.	32
4.2	Example input for the pipeline containing multiple hand-drawn characters.	33
4.3	Output of the Mask R-CNN [HGDG17] showing bounding boxes, classification and mask for each detected object.	35
4.4	Residual learning building block adapted from He et al. [HZRS16].	35
4.5	(a) Sketch example of the SketchParse dataset. (b) Annotation of the sketch where each colour corresponds to a body part. [SDBM17]. Blue denotes the head, red the torso and green the limbs. (c) Annotation transformed to a single label for training the detection step.	36
4.6	Input for the training step using multiple randomly rotated and placed sketches of the SketchParse dataset [SDBM17].	37

4.7	(a) Input containing a hand-drawn character. (b) The predicted bounding box (in red) of a detected character. (b) Predicted mask of a detected character inside the bounding box. (c) Semantic segmentation of a character with limbs (green), head (blue) and torso (red).	38
4.8	SketchParse architecture proposed by Sarvabevabhatla et al. [SDBM17].	39
4.9	(a) Binarized input, black defines 1 and white defines 0. (b) Contour (green) of the character computed using the approach by Suzuki and Abe [SA85].	40
4.10	Neighbourhood of the pixel P_1 [ZS84].	41
4.11	(a) Skeletonised character with bones (white) and joints and end-effectors (red = type <i>body</i> , blue = type <i>head</i> , green = type <i>limb</i>) (b) Heuristically simplified skeleton by removing end-effectors of type <i>head</i> and <i>body</i> if they are incident to another node of the same type.	43
4.12	(a) Skeletonised character with bones (white) and joints and end-effectors (red = <i>body</i> , blue = <i>head</i> , green = <i>limb</i>) (b) Skeleton after applying H_1 . (c) Skeleton after applying H_2	44
4.13	Red denotes <i>body</i> vertices, green for <i>limb</i> vertices, yellow for <i>wing</i> vertices, orange for <i>tail</i> vertices and blue for <i>head</i> vertices. (a) Reference skeleton for a humanoid character. (b) Reference skeleton for a quadruped character. (c) Reference skeleton for a flying character. (d) Skeleton obtained from the previous step. (e-h) Show diverse example shapes that are used to generate the skeletons.	46
4.14	Examples for two skeleton paths marked in green and orange on the skeleton denoted in white. Pink indicates a sample point on the skeleton path with its associated maximal disc centred on the sample point while still contained inside the shape.	49
4.15	(a) Distance transform computed for each pixel on the shape of the character. Lower intensity indicates a higher distance to the boundary of the shape. (b) Uniformly distributed points (black) on the shortest skeleton path connecting two end-effectors. Turquoise lines represent the skeleton obtained by the previous step.	50
4.16	Mapping (pink) of the end-effectors of the reference skeleton (left) and the detected hand-drawn character (right). Lines in blue show the actual skeleton lines, while black shows the bones (simplified) connecting the joints and end-effectors.	53
4.17	Four cases of the marching squares algorithm. + and - indicate whether the corner of the voxel is inside or outside the shape, while the squares represent the voxels. Red-marked are new edges and vertices added in each case.	54
4.18	(a) Contour of the shape determining the area that should be triangulated. (b) Triangulation using Voronoi regions shows undesired thing triangles. (c) Quad mesh using a voxel-based approach.	55
4.19	(a) Final 2.5D model generated from a detected hand-drawn character. (b) The animated character using retargeted animation of the reference skeleton.	57
		79

6.1	Six selected consecutive (from left to right and top to bottom) frames of the example movement applied to the hand-drawn monkey character.	64
6.2	Six selected consecutive (from left to right and top to bottom) frames of the example movement applied to the hand-drawn cow character.	64
6.3	Six selected consecutive (from left to right and top to bottom) frames of the example movement applied to the hand-drawn giraffe character.	65
6.4	Six selected consecutive (from left to right and top to bottom) frames of the example movement applied to the hand-drawn donkey character.	65
6.5	Six selected consecutive (from left to right and top to bottom) frames of the example movement applied to the hand-drawn bunny character.	66
6.6	Six selected consecutive (from left to right and top to bottom) frames of the example movement applied to the hand-drawn panda character.	66
6.7	6 selected frames of the example movement applied to the hand-drawn fish character.	67
6.8	(a) Shows a simple stickfigure character. (b) After detection and identification as a humanoid character, the end-effectors are mapped correctly to the prototype skeleton.	67
6.9	Shows the limitation if separate limbs are detected but the meshing does not consider them to be separate.	72
6.10	(a) Shows a character's skeleton end-effectors (right) wrongly assigned to the reference skeleton (left). (b) Shows the panda character's skeleton end-effectors (right) correctly assigned to the reference skeleton (left).	73

List of Tables

5.1	Shows the runtime of each pipeline step in milliseconds (ms).	59
6.1	Table shows selected results of the proposed approach. A ✓ indicates the success of the given pipeline step, while a ✗ indicates that this pipeline step failed. Inputs marked with † indicate that they are taken from the SketchParse dataset of Sarvadevabhatla et al [SDBM17].	69
6.2	Table shows selected results of the proposed approach. A ✓ indicates the success of the given pipeline step, while a ✗ indicates that this pipeline step failed. Inputs marked with † indicate that they are taken from the SketchParse dataset of Sarvadevabhatla et al [SDBM17].	70
6.3	Table shows selected results of the proposed approach. A ✓ indicates the success of the given pipeline step, while a ✗ indicates that this pipeline step failed.	71



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Algorithms

4.1	Thinning algorithm by Zhang et al. [ZS84]	41
4.2	Ordering nodes clockwise based on their angle.	51
4.3	Compares two end-effectors based on their angle towards a given centre.	52
4.4	Weight assignment to vertices of the generated mesh.	56



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Bibliography

- [ADV20] Martin Andersen, Joachim Dahl, and Lieven Vandenberghe. Cvxopt: Convex optimization. *Astrophysics Source Code Library*, 2020.
- [ALCS18] Andreas Aristidou, Joan Lasenby, Yiorgos Chrysanthou, and Ariel Shamir. Inverse kinematics techniques in computer graphics: A survey. In *Computer graphics forum*, volume 37, pages 35–58. Wiley Online Library, 2018.
- [AMAZ17] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 international conference on engineering and technology (ICET)*, pages 1–6. IEEE, 2017.
- [AY15] Oksana V Anikina and Elena V Yakimenko. Edutainment as a modern technology of education. *Procedia-Social and Behavioral Sciences*, 166:475–479, 2015.
- [BL08] Xiang Bai and Longin Jan Latecki. Path similarity skeleton graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(7):1282–1292, 2008.
- [BN06] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.
- [BPRS23] Benjamin Beltzung, Marie Pelé, Julien P Renoult, and Cédric Sueur. Deep learning for studying drawing behavior: A review. *Frontiers in Psychology*, 14:992541, 2023.
- [Bun97] Horst Bunke. On a relation between graph edit distance and maximum common subgraph. *Pattern recognition letters*, 18(8):689–694, 1997.
- [Can86] John Canny. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, (6):679–698, 1986.
- [CBHK04] German KM Cheung, Simon Baker, Jessica Hodgins, and Takeo Kanade. Markerless human motion transfer. In *Proceedings. 2nd International Symposium on 3D Data Processing, Visualization and Transmission, 2004. 3DPVT 2004*, pages 373–378. IEEE, 2004.

- [CGK⁺09] Otfried Cheong, Joachim Gudmundsson, Hyo-Sil Kim, Daria Schymura, and Fabian Stehn. Measuring the similarity of geometric graphs. In *SEA*, pages 101–112. Springer, 2009.
- [Com18] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018.
- [DBVKOS97] Mark De Berg, Marc Van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry: Introduction*. Springer, 1997.
- [Den12] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.
- [Dij22] Edsger W Dijkstra. A note on two problems in connexion with graphs. In *Edsger Wybe Dijkstra: His Life, Work, and Legacy*, pages 287–290. ACM New York, NY, USA, 2022.
- [DKT98] Tony DeRose, Michael Kass, and Tien Truong. Subdivision surfaces in character animation. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 85–94, 1998.
- [DSC⁺20] Marek Dvorožňák, Daniel Šykora, Cassidy Curtis, Brian Curless, Olga Sorkine-Hornung, and David Salesin. Monster mash: a single-view approach to casual 3d modeling and animation. *ACM Transactions on Graphics (ToG)*, 39(6):1–12, 2020.
- [Fol96] James D Foley. *Computer graphics: principles and practice*, volume 12110. Addison-Wesley Professional, 1996.
- [For86] Steven Fortune. A sweepline algorithm for voronoi diagrams. In *Proceedings of the second annual symposium on Computational geometry*, pages 313–322, 1986.
- [Fuk75] Kunihiko Fukushima. Cognitron: A self-organizing multilayered neural network. *Biological cybernetics*, 20(3-4):121–136, 1975.
- [Gir15] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.
- [HG DG17] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.
- [HKS17] Daniel Holden, Taku Komura, and Jun Saito. Phase-functioned neural networks for character control. *ACM Transactions on Graphics (ToG)*, 36(4):1–13, 2017.

- [Hoa62] Charles AR Hoare. Quicksort. *The Computer Journal*, 5(1):10–16, 1962.
- [HRL10] Hsiu-Mei Huang, Ulrich Rauch, and Shu-Sheng Liaw. Investigating learners’ attitudes toward virtual reality learning environments: Based on a constructivist approach. *Computers & Education*, 55(3):1171–1182, 2010.
- [HSSC08] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.
- [HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [JSMH12] Eakta Jain, Yaser Sheikh, Moshe Mahler, and Jessica Hodgins. Three-dimensional proxies for hand-drawn characters. *ACM Transactions on Graphics (ToG)*, 31(1):1–16, 2012.
- [Jun] Junya Yamada, Yoshihiro So, Takashi Okada, Comomo Kanayama, Tatsuya Kida, Takanobu Izikawa, Rika Ono, Keinchi Seki. AR app that brings your doodle to life. <https://whatever.co/work/rakugakiar/>. Accessed: June 17, 2023.
- [Kel19] John D Kelleher. *Deep learning*. MIT press, 2019.
- [KL70] Brian W Kernighan and Shen Lin. An efficient heuristic procedure for partitioning graphs. *The Bell system technical journal*, 49(2):291–307, 1970.
- [KM15] Korina Katsaliaki and Navonil Mustafee. Edutainment for sustainable development: A survey of games in the field. *Simulation & Gaming*, 46(6):647–672, 2015.
- [Kor23] Thorsten Korpitsch. Github repository. <https://github.com/AnimaNoProject/semantic-aware-animation-of-hand-drawn-characters>, 09 2023. Accessed: 9 September, 2023.
- [KW20] Midori Kitagawa and Brian Windsor. *MoCap for artists: workflow and techniques for motion capture*. CRC Press, 2020.
- [LD12] Binh Huy Le and Zhigang Deng. Smooth skinning decomposition with rigid bones. *ACM Transactions on Graphics (ToG)*, 31(6):1–10, 2012.
- [Lea97] Machine Learning. Tom mitchell. *Publisher: McGraw Hill*, 1997.

- [LGK19] Maximilian Langer, Aysylu Gabdulkhakova, and Walter G Kropatsch. Non-centered voronoi skeletons. In *Discrete Geometry for Computer Imagery: 21st IAPR International Conference, DGCI 2019, Marne-la-Vallée, France, March 26–28, 2019, Proceedings 21*, pages 355–366. Springer, 2019.
- [LWKTM07] Longin Jan Latecki, Qiang Wang, Suzan Koknar-Tezel, and Vasileios Megalooikonomou. Optimal subsequence bijection. In *Seventh IEEE International Conference on Data Mining (ICDM 2007)*, pages 565–570. IEEE, 2007.
- [Map03] Carsten Maple. Geometric design and space planning using the marching squares and marching cube algorithms. In *2003 international conference on geometric modeling and graphics, 2003. Proceedings*, pages 90–95. IEEE, 2003.
- [MHLC⁺22] Lucas Mourot, Ludovic Hoyet, François Le Clerc, François Schnitzler, and Pierre Hellier. A survey on deep learning for skeleton-based human animation. In *Computer Graphics Forum*, volume 41, pages 122–157. Wiley Online Library, 2022.
- [Mit07] Tom Michael Mitchell. *Machine learning*, volume 1. McGraw-hill New York, 2007.
- [OI92] Robert L Ogniewicz and Markus Ilg. Voronoi skeletons: theory and applications. In *CVPR*, volume 92, pages 63–69, 1992.
- [Oki12] Eiji Oki. Glpk (gnu linear programming kit). In *GLPK (GNU Linear Programming Kit)*, 2012.
- [ON15] Keiron O’Shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.
- [PASA21] Dzulfikri Pysal, Said Jadid Abdulkadir, Siti Rohkmah Mohd Shukri, and Hitham Alhussian. Classification of children’s drawing strategies on touch-screen of seriation objects using a novel deep learning hybrid model. *Alexandria Engineering Journal*, 60(1):115–129, 2021.
- [PGM⁺19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, and Luca Antiga. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.
- [Pub] The Crankshaft Publishing. what-when-how skeletal animation. <http://what-when-how.com/advanced-methods-in-computer-graphics/skeletal-animation-advanced-methods-in-computer-graphics-part-2>. Accessed: 12.08.2023.

- [PZ11] Junjun Pan and Jian J Zhang. *Sketch-based skeleton-driven 2D animation and motion capture*. Springer, 2011.
- [RFC02] Céline Rémi, Carl Frélicot, and Pierre Courtellemont. Automatic analysis of the structuring of children’s drawings and writing. *Pattern Recognition*, 35(5):1059–1069, 2002.
- [RGW20] Renata G Raidou, M Eduard Gröller, and Hsiang-Yun Wu. Slice and dice: A physicalization workflow for anatomical edutainment. In *Computer Graphics Forum*, volume 39, pages 623–634. Wiley Online Library, 2020.
- [Rie15] Kaspar Riesen. Structural pattern recognition with graph edit distance. *Advances in computer vision and pattern recognition*, 2015.
- [RP66] Azriel Rosenfeld and John L Pfaltz. Sequential operations in digital picture processing. *Journal of the ACM (JACM)*, 13(4):471–494, 1966.
- [SA85] Satoshi Suzuki and Keiichi Abe. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30(1):32–46, 1985.
- [SDBM17] Ravi Kiran Sarvadevabhatla, Isht Dwivedi, Abhijat Biswas, and Sahil Manocha. Sketchparse: Towards rich descriptions for poorly drawn sketches using multi-task hierarchical deep networks. In *Proceedings of the 25th ACM international conference on Multimedia*, pages 10–18, 2017.
- [Ser23] Serokell. Introduction to convolutional neural networks. <https://serokell.io/blog/introduction-to-convolutional-neural-networks>, 07 2023. Accessed: 8 July, 2023.
- [SGXT20] Soshi Shimada, Vladislav Golyanik, Weipeng Xu, and Christian Theobalt. Physcap: Physically plausible monocular 3d motion capture in real time. *ACM Transactions on Graphics (ToG)*, 39(6):1–16, 2020.
- [SHM⁺16] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, and Marc Lanctot. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.
- [SKRW22] Marwin Schindler, Thorsten Korpitsch, Renata Georgia Raidou, and Hsiang-Yun Wu. Nested papercrafts for anatomical and biological edutainment. In *Computer Graphics Forum*, volume 41, pages 541–553. Wiley Online Library, 2022.
- [SSS⁺17] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai,

and Adrian Bolton. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

- [Sta] Stanford. CHARIOT Program. <https://chariot.stanford.edu/>. Accessed: July 2, 2023.
- [SWO⁺14] Philipp Stefan, Patrick Wucherer, Yuji Oyamada, Meng Ma, Alexander Schoch, Motoko Kanegae, Naoki Shimizu, Tatsuya Kodera, Sebastien Cahier, and Matthias Weigl. An AR edutainment system supporting bone anatomy learning. In *2014 IEEE Virtual Reality (VR)*, pages 113–114, Minneapolis, MN, USA, 2014. IEEE.
- [SZKS19] Sebastian Starke, He Zhang, Taku Komura, and Jun Saito. Neural state machine for character-scene interactions. *ACM Transactions on Graphics (ToG)*, 38(6):209–1, 2019.
- [SZL⁺23] Harrison Jesse Smith, Qingyuan Zheng, Yifei Li, Somya Jain, and Jessica K Hodgins. A method for animating children’s drawings of the human figure. *ACM Transactions on Graphics (ToG)*, 42(3):1–15, 2023.
- [TBVDP04] Matthew Thorne, David Burke, and Michiel Van De Panne. Motion doodles: an interface for sketching character motion. *ACM Transactions on Graphics (ToG)*, 23(3):424–431, 2004.
- [TP23] Telecom-Paristech. Skeleton-based deformations. https://perso.telecom-paristech.fr/jthiery/cours/ig3da/docs/deformation-2/02_skeletal_deformations.pdf, 7 2023. Accessed: 2023-07-03.
- [TPP⁺22] Xien Thomas, Larry Powell, Seth Polsley, Samantha Ray, and Tracy Hammond. Identifying features that characterize children’s free-hand sketches using machine learning. In *Interaction Design and Children*, pages 529–535, 2022.
- [VGO⁺20] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

- [WNSV19] Hsiang-Yun Wu, Martin Nöllenburg, Filipa L Sousa, and Ivan Viola. Metabopolis: scalable network layout for biological pathway diagrams in urban map style. *BMC bioinformatics*, 20:1–20, 2019.
- [WP02] Xiaohuan Corina Wang and Cary Phillips. Multi-weight enveloping: least-squares approximation techniques for skin animation. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 129–138, 2002.
- [ZLZ⁺16] Hua Zhang, Si Liu, Changqing Zhang, Wenqi Ren, Rui Wang, and Xiaochun Cao. Sketchnet: Sketch classification with web images. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1105–1113, 2016.
- [ZS84] Tongjie Y Zhang and Ching Y. Suen. A fast parallel algorithm for thinning digital patterns. *Communications of the ACM*, 27(3):236–239, 1984.
- [ZSKS18] He Zhang, Sebastian Starke, Taku Komura, and Jun Saito. Mode-adaptive neural networks for quadruped motion control. *ACM Transactions on Graphics (ToG)*, 37(4):1–11, 2018.