TECHNISCHE UNIVERSITÄT
CHEMNITZ

BACHELOR THESIS

# Implementation of a modular pipeline to evaluate different rigging and retargeting techniques for virtual humans using CrossForge

Faculty of Computer Science
Professorship of Computer Graphics and Visualization

*Author:*
Mick KÖRNER

*Examiner:*
Prof. Dr. Guido BRUNNETT
*Supervisor:*
Dr.-Ing. Thomas KRONFELD

*A thesis submitted in fulfillment of the requirements*
*for the degree of Bachelor of Science*

January 3, 2025

UNIVERSITY OF TECHNOLOGY CHEMNITZ

# *Abstract*

Professorship of Computer Graphics and Visualization

Bachelor of Science

**Implementation of a modular pipeline to evaluate different rigging and retargeting techniques for virtual humans using CrossForge**

by Mick KÖRNER

The Thesis Abstract is written here (and usually kept to just this page). The page is kept centered vertically so can expand into the blank space above the title too…

# *Acknowledgements*

# Contents

# List of Figures

# List of Tables

# Notes

parts starting with "-" need to be rewritten

# Chapter 1

# Introduction

## 1.1 Motivation

Virtual Humans have been a major Part of Computer Graphics because of its wide range applications, spanning multiple research domains.
Creating a realistic Virtual Human is still a challenge today. Digital Reconstruction techniques like Strucute-from-Motion can create a very Detailed Surface replication of a Person. However, this Mesh is static. If it is desired to animate this Scan with Motion Capture Data, the Mesh does not contain any Information on how to apply these.
While Motion-Capture techniques like Shape-from-Silhouette exist, which are creating an Animation by storing a 4D Mesh. The use Cases for these Results are limited because the Motion and Virtual Character are coupled.

Simplifying the Virtual Human problem to decouple Motion- and Surface Data has naturally developed to be the standard today, not only for Realistic Virtual Humans, but also heavily stylized ones in Movies and Games.

TODOm 1.1.1: Motiovation or Objectives and Scpoe?

- Another important Motivation was to provide an easy to access and open source tool for motion retargeting, all widely used retargeting tools either require payment or an account login. Notibly there do no exist solid free motion retargeting Solutions.

- no basic tool for simple customizable motion retargeting

- while ik is already a common tool for animators to quickly get a desired pose, a well implemented and accessable motion retargeting can further improve an animators workflow by posing as a starting base for a desired pose using other motion editing tools

A deeper look into existing tools for these Problems reveals that many of them are sub-optimal or require some form of payment. Either in form of Currency or User Data.

## 1.2 Objectives and Scope

To facilitate the option to use a large set of Motion Data with Rigged Characters popular Tools like Mixamo use standardized Human like Skeleton to simplify the Process by moving the Motion Retargeting Problem to a Auto-Rigging Problem. Thus for a scalable system, the underlying Skeleton should be abstractable and independent of Motion Data. This is however not easy.

The primary Goal is a Tool which automates or streamlines the process of creating a Virtual Character just from a Scan. This includes the Implementation of Interfaces to easily add new methods for Autorigging and Motion Retargeting.

To further support Scalability for Future use. The proposed Tool should be interactive in order to test and compare algorithms more easily for correctness and potential drawbacks.

## 1.3 Summary of the Work

Firstly we will go over all Related Works in Chapter 2. This includes a Recap of how Computer Animation works and their basics. Then we go over Inverse Kinematics, Constraints up to Motion Retargeting and AutoRigging in Chapter 2.

In each Chapter, fundamentals are explained. Popular and recent novel techniques will be discussed and available tools will be evaluated, not primairly in performance but also in availability, open-ness and ease of use.

In Chapter 3 the Design and Implementation of the Automation Tool is explained. As well as details about specific Implementations of Motion Retargeting and Autorigging Methods or API interfaces.

# Chapter 2

# Related Work

## 2.1 3D Animation Fundamentals

Prior to examining the literature pertinent to this thesis, it is essential to define the fundamental principles of skeletal animation in computer graphics, establish consistent nomenclature, and establish a foundation to prevent confusion. In the field of cross-paper naming, it is not uncommon for different designations to be used for the same concept or for separate concepts to be merged into a single term.

In addition, many papers adopt a clear and consistent naming convention prior to review.

The most prevalent form of humanoid animation is skeletal animation. The majority of graphics engines are capable of supporting this type of animation due to its inherent simplicity. This has led to its early adoption as a standard feature in hobby engines, with numerous motion editing tools in the industry also built around it.

### 2.1.1 Skeletal Animation

- simillar to how a animals in the real world have a rigid bones connected to a skeleton and moved with muscles, a similar analogy developed in computer graphics in a bionics manner

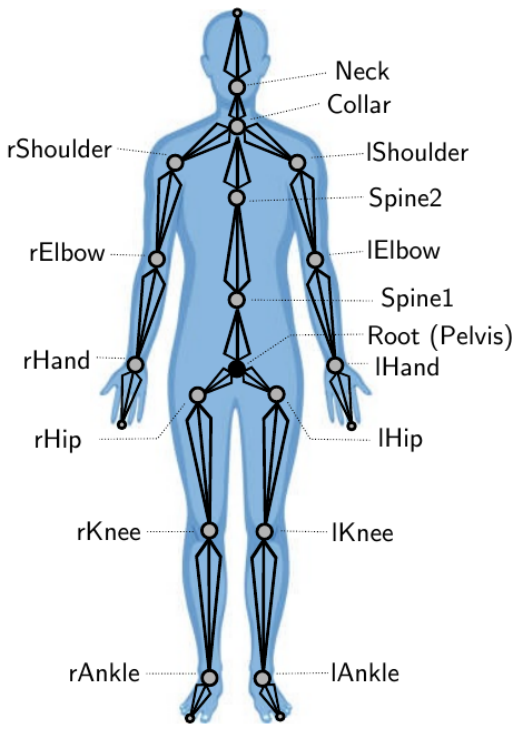

FIGURE 2.1: Example of human skeleton, note that bones and their parent joint are combined, this can cause confusion, in this example the root and collar joint have multiple bones. Image taken from [1]

Bones (sometimes called links) in Animation represent rigid objects inside virtual character - associated with a length attribute.

Joints represent the connection points between bones and are characterized by up to three by up to three rotational (Degree of Freedom) DoF. - joint is the component concerned with motion;

In addition to joints connecting two or more bones, root and end effector joints are of particular interest.

A root joint has no parent. Any transformation applied to this joint is reflected in the actor's global movement. In animation, this joint is often translated in conjunction with a walking animation, ensuring that the actor does not remain stationary while walking. While this could be achieved through the use of a scenegraph, it facilitates the unification of motion playback across applications by circumventing the necessity for an additional abstraction.

- endeffectors represent bones without children - depending on application sometimes it isnt clear if the endeffector is joint itself or a bone, having no joint at its tip, due to these discrepencies some systems having additional joints defined at it tips to ensure conversion between different formats happen seamlessly

- Bones are usually not explicitly defined in implementations and are implicitly included in their parent joint

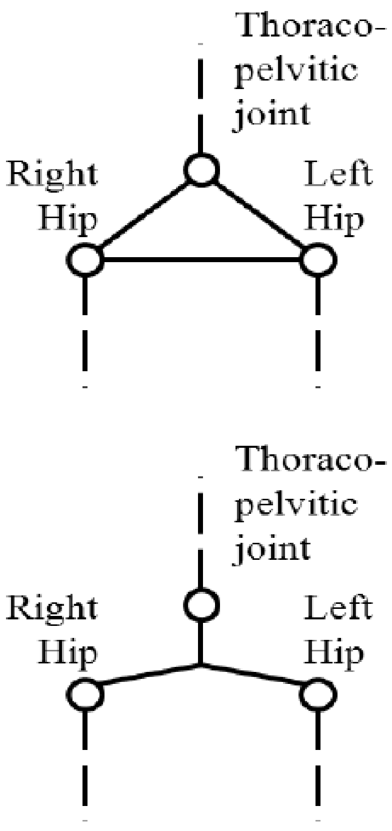

FIGURE 2.2: Visualization of an Armature with a loop, depending on implementation the tree becomes a Graph. Image taken from [2]

### 2.1.2 Skeletal Hierarchy

A skeleton is comprised of multiple bones arranged in a hierarchical structure, typically a tree-like configuration.

explain chain

- a joint chain represents a link of multiple joints where each joint has at most 1 child

- branching happens when a joint has more than one child

TODOm 2.1.1: visualize? + position in TEX?

chain loops, extend

Closed loops - while tree structures are most often found, some systems allow for circular structures - using smartly placed bones, one can enforce constraits, for example ensuring 2 bones have always - harder to implement

- In implementations Bones and their parent joints are often combined. Since the parent joint describes the rotation of th

meaning

? - useful for centre of rotation correction - not allowed in some systems, e.g. blender

### 2.1.3 Pose Space vs. Work Space

Established common Spaces in the Graphics Pipeline include Window Mapping (NDC and Camera space), but more importantly for this work, World Space and Object Space. Object Space in regards to Skeletal Animation means the Space of the character in restpose.

later important expl

- In order to visualize a skeleton or parent other objects in worldspace to joints, for example a tool to simulate some kind of work. We need to know the position of a desired Joint in pose $\theta$.

As discussed previously joints describe rotation of their child bones. To determine Position of Joints relative to Object Space, all kinematic chains from the root bone have to be propagated.

### 2.1.4 Forward Kinematics

- forward kinematics describes the process of computing the working space from pose space parameters Let $F$ be the forward Propagation of the kinematic chain and $\theta$ the current pose configuration, object space position and rotation $t$ of the endeffector can be computed as:

$$T = F(\theta)$$

2.3 visualizes this process, translation, rotation (and sometimes scale) of each joint determines the transformation of all child joints.

math formula example

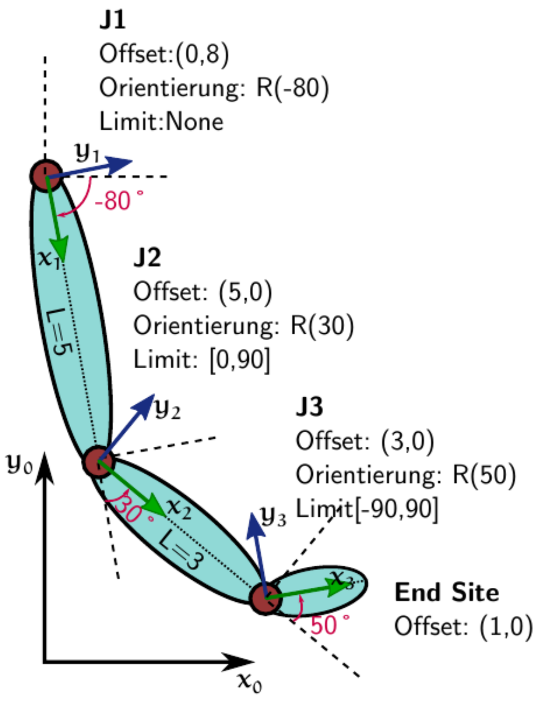

FIGURE 2.3: Example of a joint chain with respective local coordinate systems visualized, notice that the global transform of Joint J2 depends on J1, and J3 on J2 and J1. Image taken from [1]

explain affine matrix multiplication (rotation + translation)

- for affine transformation the propagating the chain results in object space relative rotation and translation

### 2.1.5 Restpose and Bind Pose Matrix

- Because Character Modellers or Scans have to define the surface of a Virtual Character in an existing pose, Bones have to be placed correctly in that Character. Joint rotations of a Motion are then applied relative to the restpose angle of that joint.

This also suggest that motion transfer between skeletons poses already a challenge when restposes are different.

explain what term rig mean beforehand

The Bind Pose Matrices are assigned per Joint and describe the transformation from the Object Space Koordinate system of the rigged character to the corresponding Joint in Restpose.

The Inverse Bind Pose Matrix, as the name implies, does the opposite of the bind pose matrix, in various paper and code sources this is also commonly referred to as Offset Matrix.

Both Bind Pose and Offset matrix are defined with the skeletal hierarchy and their restpose once for a character. The Offset Matrix is essential part for efficient Linear Blend Skinning.

### 2.1.6 Skeletal Skinning

- for now we have a skeletal definition, but what was initialy wanted was to animate a character mesh easily - the Idear of Skeletal Animation is to abstract parts of the body away into joints, this is to reduce the complexity by defining motion of every single surface vertex manually. For Skeletal Animation, Vertices of the character surface, also called Skin, is abstracted to a bone.

This is done by assigning which vertex is affected by which bone. Furthermore, because Flesh is deformable and not rigid, there is a need to interpolate vertices near the joint of two bones, for a 2 bone example and a vertex inbetween them.

- depending on what kind of cloth a character is wearing, there is a need to define vertex weights. Vertex weights have been hand authored by weight painting or tools like

blender automatic weight computation, nearest bone name

- The most common used Skinning method is Linear Blend Skinning - there are many more skinning methods which try to fix artefacts of linear blend skinning, but this is not in the scope of this thesis

TODOm 2.1.2: in CForge?

- for linear blend skinning, the offsetmatrix moves the weighted vertices of a joint in object space to the center of the coordinate system, so that local rotations of a joint are applied correctly. Together the joint transformation chain with the offset matrix are combined into the skinning matrix, which is then applied in the vertex shader during rendering.

explain math rotation and translation

TODOm 2.1.3: unnecessary? or later?

- Box-Based or Spherical Skinning - Dual Quaternion Skinning (DQS) - Delta Mush

### 2.1.7 Motion Data

For Motion Playback, Rotational, Translation and Scale values, per Joint. One pose configuration in an Animation is called Keyframe. A Motion consists of multiple keyframes played sequentially. Timepoints per Keyframe determine at which time of an Animation a given Pose should be displayed.

The Sampling rate determines how many Keyframes per second are contained in the animation.

- a common trick for gait motion is to use the sampling rate to create a variable amount of walking speeds from one animation without having to create or capture gait motion for every desired speed - nearest neighbor interpolation between keyframes would result in choppy animation playback, to get a smooth playback at lower sampling rates linear interpolation is an quick, ease and sufficient enough for pleasing results

### 2.1.8 Other Animation Approaches

- Morph Targets - vertex animation textures (Vertex Shader Animations)
-> no one as flexible as skeletal animation due to abstraction of skeletal animation

## 2.2 Inverse Kinematics

check

- forward kinematics desribed at we have joint angles and lengths, with which we can compute each subsequent joint starting point to get the endeffector position - inverse kinematics describes the need to get joint angles with which rigid joint lengths and a target position, the endeffector matches the target position

In the preceding section, it was established that forward kinematics processes inputs from the configuration space of a rigged model, yielding working space coordinates that can be utilized for rendering a skinned mesh. Additionally, collision tests can be conducted, and further objects can be parented to joints.

For an dynamic grabing motion a natural desire would be to know a Configuration to target any Point in Working Space.

formulation

- Definition IK - ik goal to find joint configuration where endeffectors move to desired targets, while movement should be smooth fast and accurate

Inverse kinematics (IK) is the process of determining a joint configuration that satisfies various working space conditions, such as reaching a target or avoiding specific regions in space.

formulation

- Animators use Inverse Kinematics to intuitively animate characters without having to rotate each bone individually

Inverse Kinematics pose a fundamental tool for Motion Editing, its not only used for Automating Processes or real time interactive applications, but by 3D animators themselfs as a helpful tool to model a desired pose more easily and quickly.

formulation

- very useful in animation such as movies and games as well as robotics - Inverse Kinematics widely used in Animation and Robotics industry

### 2.2.1 The IK Problem

reuse explaination of basics

The ideal approach would be to find a inverse mapping of the Forward Kinematics Mapping $F$ so can get a pose configuration $\theta$ for a given target direction $t$:

$$\theta = F^{-1}t$$

- The primary challenge associated with inverse kinematics lies in the fact that pose space and working space are not linearly dependent.

### 2.2.2 Reachability

- This is beacause the Inverse Kinematics Problem can not be solved unambiguously. - figure 2.4 shows that in 2D a chain of more than 2 joints yields an infinite amount of solutions for an reachable point

cases

- 3 cases - target is outsite of reach, no solution - depending on desired behavior, chain should be - target is at distance of of chain length ik is applied to - exactly one solution - easily identifyable, but rare occurance

- target is within bounds of chain length - still problem that depending on skeletal definition sometimes points inside chain length are not reachable (e.g. long and small joint)

In 2D, a chain of more than two joints yield an infinite amount of solutions for a reachable Point in space, illustrated in Figure 2.4. This already happens in 3d for chain length of two.

2D example infinite solution

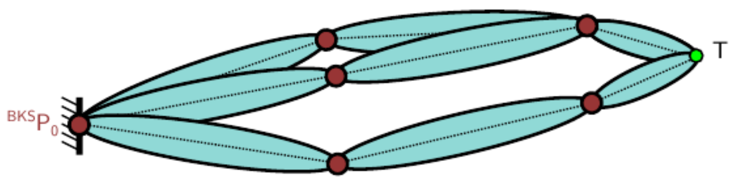

FIGURE 2.4: In 2D, for Chains with more than 2 Joints, there will be an infinite Amount of Configurations satisfying reaching the Target, when $|T - P_0| < \sum_i |j_i|$. Image taken From [1]

- multiple solution if chain length == target distance to chain root 1 sol - if target outside, no solutions

[3] has expl

### 2.2.3 Analytical Methods

- The analytical approach tries to solve the system of equations spanned by inverting the Forward Kinematics formula of the corresponding armature. Because they find solutions reliably, they are called Closed form solutions.

- Lander [4] explained the analytical method simple for beginners.

main expl

- Figure 2.5 illustrates relevant Variables for solving a 2 Joint chain. $l_T$, $l_1$ and $l_2$ span a triangle, because all lengths are given, trigonometric functions can be used to determine angles $\theta_1$ and $\theta_2$.

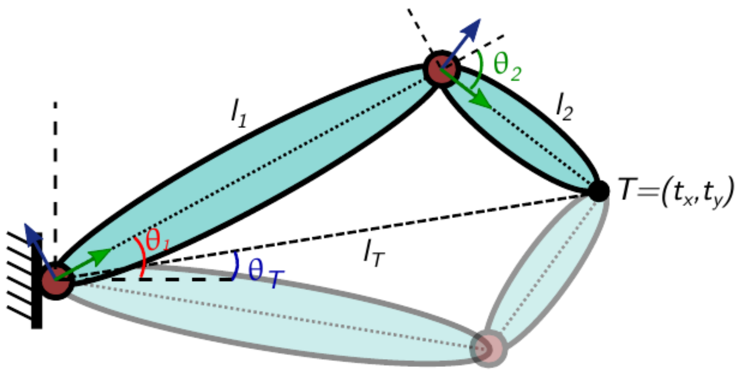

FIGURE 2.5: Visualization of relevant Variables for solving $\theta_1$ and $\theta_2$ analytically to reach point $T$ using Trigonomic functions. A Second Solution is Visualized with less opacity below, Image taken from [5]

While this solution would be Ideal because it is very fast and numerically stable. Solving the system for more than two Joints becomes with each additional Joint more complex.

- but the computational increase with longer chains is not the only problem. providing either all Solutions or predetermined ones, finding solutions that produce plausible results related to temporal locality is even more difficult

### 2.2.4 Jacobian Methods

The Jacobian Inverse Method for solving Inverse Kinematics falls into the category of numerical solvers and represents the first Iterative Approach developed.

- also called inverse rate control

check

TODOm 2.2.1: formulation

Previously 2.2.1 the Problem of multiple pose space configurations satisfying target position constraints. This implies that there exist multiple mappings of $F^-1$ that could potentially satisfy $t$. Consequently, determining the optimal solution becomes a complex task.

Furthermore, it is not uncommon for $F$ to lack direct invertibility. This further complicates the determination of a unique and well-defined inverse function, or even the existence of such a function across the entire workspace.

go into detail with 2.6

When a joint is rotated, the resulting endeffector moves in a circular motion. This indicates that the forward kinematics function outputs a non-linear space in which the endeffector moves. 2.6 visualizes this difference for an endeffector.

However, it can be observed that this non-linear space can be approximated by a linear space for small amounts of movement:
Let $J$ be a linear space mapping such that for a small movement of $\theta$:

$$\Delta t \approx J(\theta)\Delta(\theta)$$

The Jacobian Matrix $J$ is defined as the rate of change on Vector $t$ when we turn angles of Joints in $\theta$ in each respective Dimension for a small amount $\Delta$.

- Explicit values of $J$ can then be evaluated by changing the corresponding angle of the armature by $\Delta$ and using the Forward Kinematics Function to determine the change of endeffector direction relative to its old position in object space.

For rate of change a common definition of the Jacobian Matrix is representing it using derivatives:

fill

$$J = \left( \frac{\partial F(\theta)_i}{\partial \theta_j} \right)$$

where $i$ are respective Dimensions in which the target moves for each changeable angle $\theta_j$.

$$\Delta\theta \approx J^{-1}(\theta)\Delta(t)$$

cite
https://www.youtube.com/watch?
and replace image with own

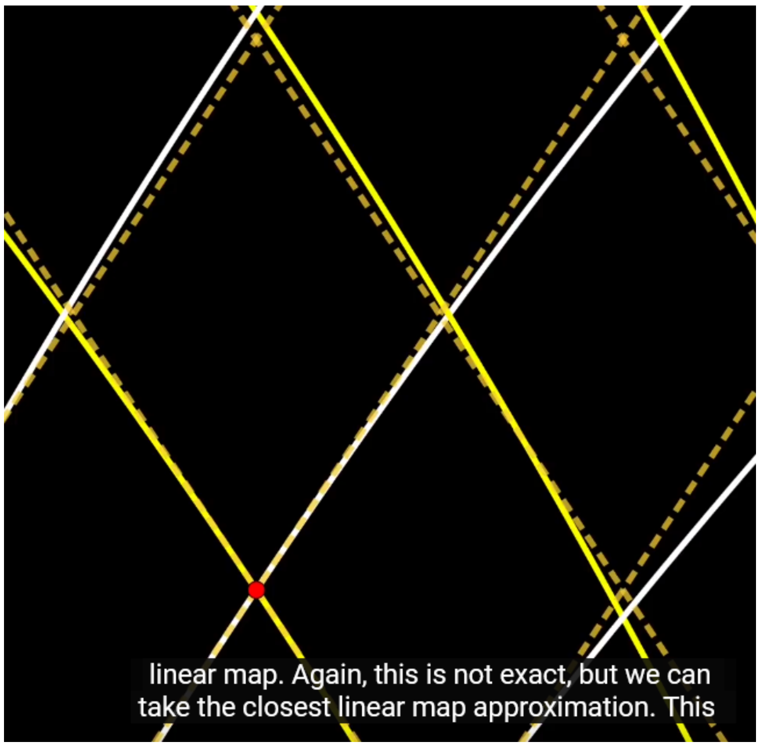
linear map. Again, this is not exact, but we can
take the closest linear map approximation. This

FIGURE 2.6

put rigid explaination of simplifying calculation into chapter 3

Buss [6] provides a more in-depth Introduction to the Jacobian Inverse Kinematics Method and how the Jacobian Inverse works.

- still lacking resources on how to implement Jacobian IK

### 2.2.5 Cyclic Coordinate Descent

Cyclic Coordinate Descent (CCD) were the first heuristic approaches to solving IK. Kenwright [7] wrote a great article which summarizes the History Workings and Constraints. There he stated that, due to its simplicity, it is not certain who published, but Wang and Chen [8] are credited.

- in order to reach a target point with an endeffector, each joint will be rotated so that the current vector from current joint position to endeffector points to the target

expl more in depth + picture

- there are two variants of CCD, one which starts rotating joints from the endeffector joint back to the root, and one that starts from the root and rotates the endeffector last

break condition?

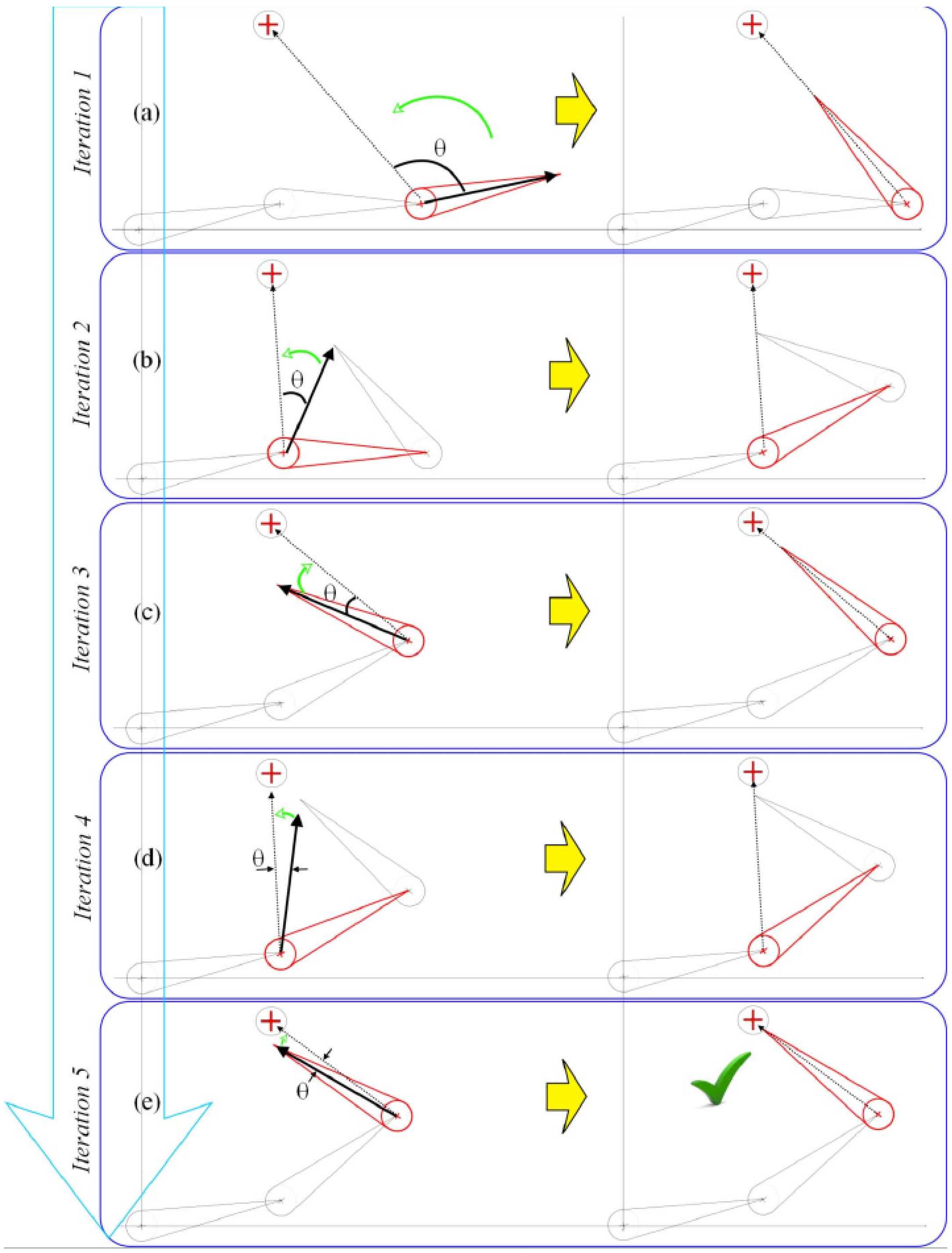

FIGURE 2.7: Three Joint Chain example of CCD, in each Iteration the Current Joint is rotated so that the Vector from current Joint to target and current Joint to endeffector align. Image taken from [7]

**Algorithm 1** BackwardCCDIK Algorithm, Taken From [7]

```
 1: procedure BackwardCCDIK
 2:     Input: $e$                                                    ▷ threshold
 3:     Input: $k_{\max}$                                             ▷ max iterations
 4:     Input: $n$                          ▷ link number (0 to numLinks-1 chain)
 5:     $k \leftarrow 0$                                              ▷ iteration count
 6:     while $k < k_{\max}$ do
 7:         for $i = n - 1$ to 0 do
 8:             Compute $u, v$                       ▷ vector $P_e - P_c$, $P_t - P_c$
 9:             Compute ang                                   ▷ using Equation 1
10:             Compute axis                                  ▷ using Equation 1
11:             Perform axis-angle rotation $(\text{ang}, \text{axis})$ of link $i$
12:             Compute new link positions
13:             if $|P_e - P_t| < e$ then                         ▷ reached target
14:                 return                                            ▷ done
15:             end if
16:         end for
17:         $k \leftarrow k + 1$
18:     end while
19: end procedure
```

### 2.2.6 FABRIK

address CCD problems in CCD sec

- In order to improve performance and the rolling and unrolling Problem of CCDs, Aristidou and Lasenby [9] came up with Forward And Backward Reaching Inverse Kinematics (FABRIK)

- builds and optimizes upon ccd - fabrik noted producing more natural results, avoiding rollung and unrolling of ccd and moving the whole chain like jacobian inverse

- fabrik simplifies the

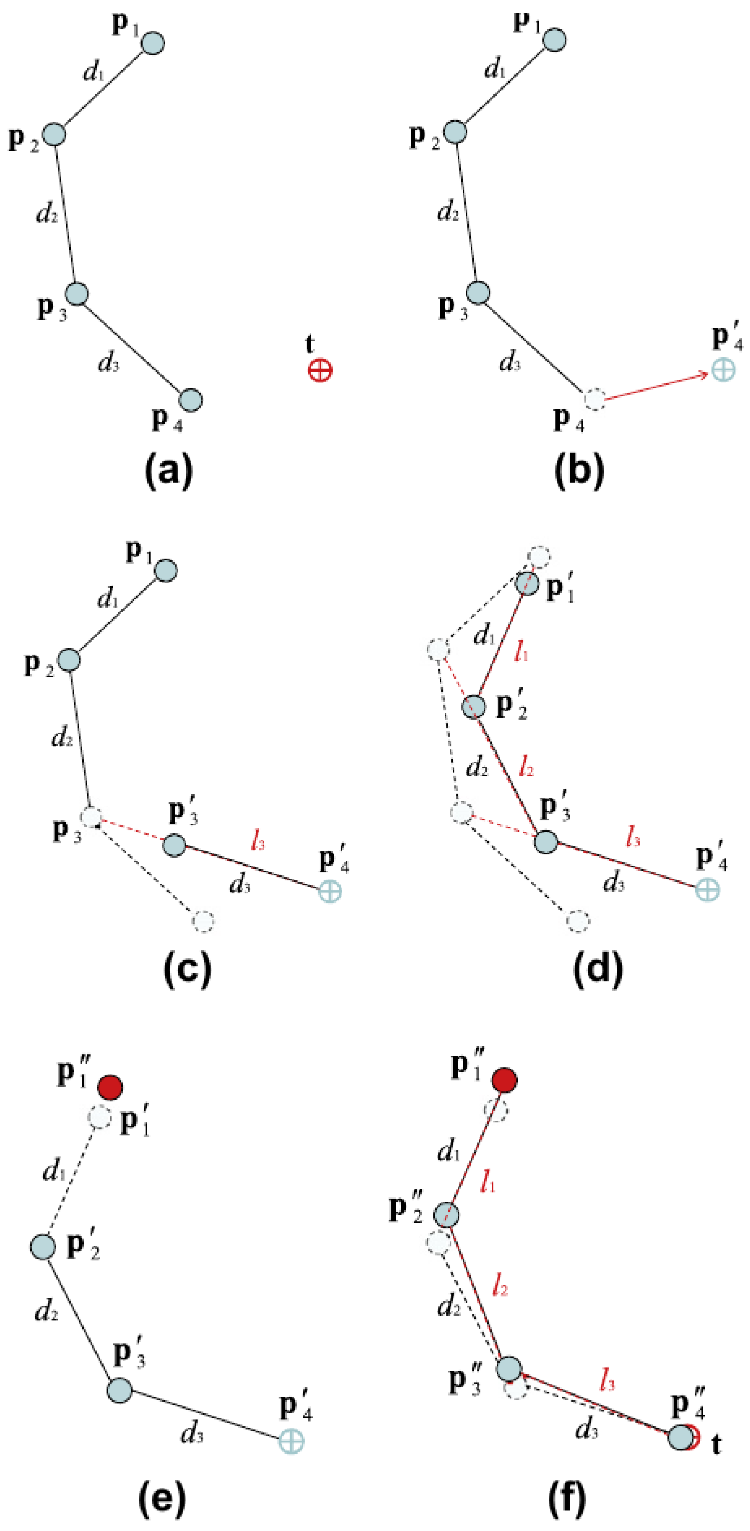

FIGURE 2.8: Image taken from [9]

**Algorithm 2** A full iteration of the FABRIK algorithm, Taken from [9]

```
Require: The Joint positions $p_i$ for $i = 1, \dots, n$,
Require: target position $t$,
Require: distances $d_i = \|p_{i+1} - p_i\|$ for $i = 1, \dots, n-1$
Output: New joint positions $p_i$ for $i = 1, \dots, n$
 1: dist $\leftarrow \|p_1 - t\|$                      ▷ The distance between root and target
                                                        ▷ Check whether the target is within reach
 2: if dist > $d_1 + d_2 + \dots + d_{n-1}$ then         ▷ The target is unreachable
 3:     for i = 1 to n-1 do     ▷ Find the distance $r_i$ between the target t and the joint
    position $p_i$
 4:         $r_i \leftarrow \|t - p_i\|$
 5:         $k_i \leftarrow \frac{d_i}{r_i}$
 6:         $p_{i+1} \leftarrow (1 - k_i)p_i + k_i t$          ▷ Find the new joint positions $p_i$.
 7:     end for
 8: else
             ▷ The target is reachable; thus, set as $b$ the initial position of the joint $p_1$
 9:     $b \leftarrow p_1$
10:     $dif_A \leftarrow \|p_n - t\|$
            ▷ Check whether the distance between the end effector $p_n$ and the target t
    is greater than a tolerance.
11:     while $dif_A > tol$ do
                                                        ▷ STAGE 1: FORWARD REACHING
12:         $p_n \leftarrow t$                             ▷ Set the end effector $p_n$ as target t
13:         for i = n-1 down to 1 do
             ▷ Find the distance ri between the new joint position $p_{i+1}$ and the joint p
14:             $r_i \leftarrow \|p_{i+1} - p_i\|$
15:             $k_i \leftarrow \frac{d_i}{r_i}$
16:             $p_i \leftarrow (1 - k_i)p_{i+1} + k_i p_i$     ▷ Find the new joint positions $p_i$ .
17:         end for
                                                        ▷ STAGE 2: BACKWARD REACHING
18:         $p_1 \leftarrow b$                             ▷ Set the root $p_1$ its initial position.
19:         for i = 1 to n-1 do
                             ▷ Find the distance $r_i$ between the new joint position $p_i$
20:             $r_i \leftarrow \|p_{i+1} - p_i\|$
21:             $k_i \leftarrow \frac{d_i}{r_i}$
22:             $p_{i+1} \leftarrow (1 - k_i)p_i + k_i p_{i+1}$   ▷ Find the new joint positions $p_i$.
23:         end for
24:         $dif_A = ||p_n - t||$
25:     end while
26: end if
```

### 2.2.7 Other Methods

There exist many more Methods for solving Inverse Kinematics, but - fail due to high cost

Newton Methods explained by [9] treat IK as a minimization problem but are slow and hard to implement.

Mass Spring Models, is a IK method proposed by Sekiguchi and Takesue [10]. A numerical method which uses the virtual spring model and damping control. Suited for redundant robots, robots which have many DOF.

DOF Degrees of Freedom explain in basics

explain mass spring

expl Particle IK

TODOm 2.2.2: "Den Inhalt würde ich am Anfang von Kapitel 2.2. einfügen. (Direkt nach der Problemformulierung)", weitere infos mit vergleichen zwischen ik schon hier?

### 2.2.8 IK Surveys

Aristidou et al. [3] present various Inverse Kinematics techniques in depth for general applications.

Boulic et al. [11] survey different Inverse Kinematics techniques to correct noisy and incomplete motion capture data from vision Input.

`https://zalo.github.io/blog/inverse-kinematics/#properties-of-various-ik-algorithms`

complete

table to visualize comparison of ik methods like

## 2.3 Constraints

- In the previous section we looked at Inverse Kinematics abstractly as a motion editing tool, because IK is dynamic in nature and only considers Skeletal strucutre

Compared to the real world, we have yet to model DOF limiting factors of our Skeleton Bones like neighboring tissue like muscles, organs, fat or Connective tissue, as well as physical limits related to the atanomy and structure of the bones themselfes located at joints.

These are essential for Inverse Kinematics and its appliences in order to already avoid a set of self interpenetration Issues as well as non plausible poses to improve realism.

- many papers describe constraints specifically for an inverse kinematics method

- integration tied to a specific inverse kinematics method allows for optimization potential

- problematic is to incorporate constraints in a way that a global solution will still be found

### 2.3.1 Tree Structures

- in order for

- multiple inverse kinematic chains that share the same joint

- Jacobian Inverse Kinematics solves this naturally by incorporating not just one chain and a target, but all joints and targets into tha jacobian matrix.

### 2.3.2 Skeletal Constraints

- Aristidou et al. [12] have described six most common anthro- pometric Joint constraints, visualized in Figure 2.9.

- dependin on tpye various types of movements allowed

ball-and-socket joint - ball moves within a socket - limits angular rotation in the direction of parent joint

- hinge joint - simplest type of joint; - elbows, knees - motion only in one plane/direction about a single axis

- pivot Joint - only rotation on one axis, used in neck - for a given target, the head orientates towards it, - the target point has to be projected on the axis, and the rotation constraint has to be enforced

condyloid - ovoid articular surface that is received into an elliptical cavity - permits biaxial movements, that is, forward-backward and side to side, but not rotation

saddle - convex-concave surface, treated same as condyloid, e.g. thumbs - different angle limits, allowable bounds - no axial rotation

plane joint - also gliding joint, only sideways/sliding movements - requires IK rule relaxiation in form of joints are not connected anymore - done by projecting target onto joint plane bounds in algo

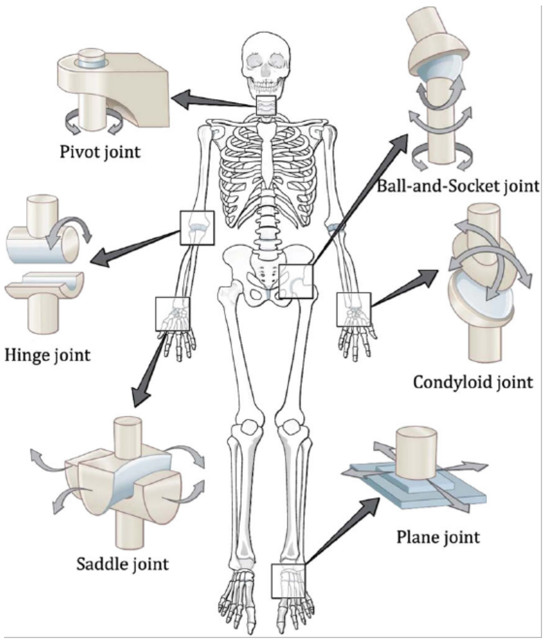

FIGURE 2.9: Various Constraint Types Visualized and where they could be used in a Virtual Human Skeleton. Image taken From [12]

### 2.3.3 Other Constraints

- Wilhelms and Gelder [13] proposed Reach cones, using spherical polygons to specifying a region for allowable joint movement.

- other constraints types that can be useful for motion editing

- distance constraints can ensure that either specific spaces are avoided or should be reached, for example elbow movement

self intersection

- the same way a constraint could be incorporated that checks for self intersection
- enforcing various constraints is difficult because it can affect an algorithms ability to converge

hinge limits

swing twist limit

### 2.3.4 Jacobian Inverse Constraints

complete

[14]

### 2.3.5 CCD Constraints

- While weighting CCD for multiple endeffectors can be intuitive Hecker [15] explained how to utilize priorities by averageing desired angles at branches.

- To ensure CCD doesnt run into a local minima under influence of constraints simulation annealing is used [7]. - Simulation annealing for CCD tries to jump out of a local minima by randomly rotating joints.

### 2.3.6 FABRIK Constraints

- [12] mentioned multiple various constraint types, but lacks in detail on how to exactly implement these, referring to [9].

- While mentioned in [9], Aristidou et al. [12] explained more in detail how to solve a FABRIK armature for multiple endeffectors.

- all chains of the armature are propagated from endeffector until the sub-base joint, which is equivilant term for branch.

Multiple Endeffectors - 2 Stages
- first Normal (forward) - appliy FABRIK starting from each endeffector moving inward - moving until sub-base: - (sub-base == joint with 2 or more child chains) - sub-base is joint that connects two or more joints - -> so apply fabrik until sub-base is reached - on subbase joint, each position of joint on fabrik iter is stored - -> centroid is calculated wich is the mean of all pos - - FABRIK iter contiued from new subbase pos (centroid) - second stage backward: - algo normal applied until sub-base - then applied seperately for each chain

multi endeff cases: - neither reachable - just apply straight line - TODOf optimized version lowers to one iteration - one reachable - 2 solutions: - attain one target, leave other away - both away but closer - TODO option to interpolate with weights - both reachable - 2 cases: - both can reach target in configuration - only one can reach target, again interpolatable with weights

caption

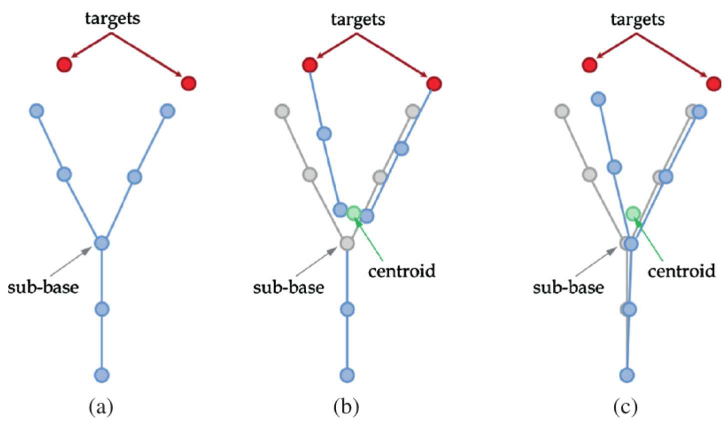

FIGURE 2.10: Image showing, Image Taken from [12]

- option to smooth out noisy input data **4.4.2. Joint Control between Two True Joint Positions.**

- true root and end eff pos, noisy inter joints - if out of reach, straight line - if in reach fabrik applied from end eff (first forward) then backward

check source

- also posibility to run into deadlock - strict constraints cause not reaching config because of locality in algo (no parent, child consid) - solution - first check if reachablie: - yes -> if dist not smaller each iter - -> backward step first iter: bend by increasing degrees away from target - allows joints to bend more - bending till 360 degrees, then no solution if not reached
- [12] also showed Self-collision Determination

### 2.3.7 iTASC

Instantaneous Task Specification and Control (iTASC) [16] - multiple constraints

iTASC blender pos

- it Implemented as an optional Inverse Kinematics Solver in Blender but incomplete, having various issues, highlighting its implementation complexity.

## 2.4 Motion Retargeting

- Motion Retargeting is the process of transfering motion data for a skeleton with a specific hierarchy and joint lenghts to another skeleton which either differs in one or both. - In the following sections various Motion Retargeting approaches of the past years are inspected - There is no clear cut categorization of what type an approach exactly is - subject to many aspects, can be categorized differently, following a categorization sensible for this work

### 2.4.1 Naive Retargeting

- The naive retargeting approach is to simply transcribe motion applied to a joint from the source character to a target character by defining joint correspondences between source and target character

list problems

- this approach can cause various problems, including but not limited to ground penetration, self interpenetration, wrong directions due to restpose differences, foot-sliding and more

### 2.4.2 Motion Cleanup Aproaches

- try to fix artifacts generated by naive approach

Tang et al. [17] presented a motion retargeting method to convert movements between characters with heterogeneous topologies. - different topology (number of nodes), different coordinates systems, and different initial posture. - manually set the joint correspondences between the two hierarchies - ignore some useless leaf nodes and nodes cannot find a corresponding

- First, adjust the initial posture of source skeleton. - but do not explain how to adjust in detail

- use adjustment matrix - Motion data from source skeleton needs to be adjusted to ensure the new skeleton has the same movement after the initial posture adjustment. The rotation of each joint is adjusted according to this initial posture adjustment.

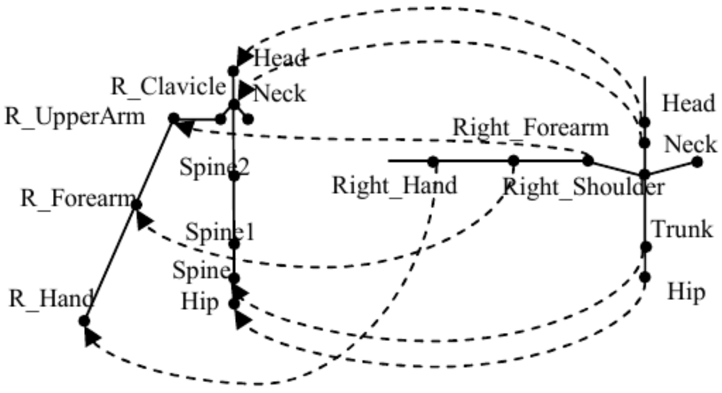

FIGURE 2.11: Image take from [17]

- Let $A_j$ be the adjustment matrix of joint $j$, and $R_j$ be the rotation matrix of joint $j$ before adjustment, Adjusted rotation $R'$:

$$\begin{aligned} R'_{L\_UpperArm} = {} & A_{Hip}A_{Plevis}A_{Spine}A_{Spine1}A_{Spine2}A_{Neck}A_{L\_Clavicle}R_{L\_UpperArm} \\ & (A_{Hip}A_{Plevis}A_{Spine}A_{Spine1}A_{Spine2}A_{Neck}A_{L\_Clavicle}A_{L\_UpperArm})^{-1} \end{aligned} \tag{2.1}$$

- Scale Root Node: The global position of the root node is scaled based on the leg length ratio between the source and target skeletons - skipping joints with no matches done with

$$R'_{\text{child}} = R_{\text{parent}}R_{\text{child}} \tag{2.2}$$

- processes on keyframes of a given animation - correct the resulting motion and make it enforce Cartesian constraints by using Inverse Kinematics

- previous section problem of defining correspondences between joints

cite smartbody source code

- Feng et al. [18] propose very similar method to Tang et al. for their Open Source Simulation Framework smartbody - Automated Skeleton Matching via similarly named joints and recognizing body part topology - use naming to determine left and right, but sometimes break at uncommon naming conventions, especially problematic because some motion capture systems only provide numeration as joint names - their offline Retargeting Process incorporates converting joint angles after aligning default poses and jacobian based inverse kinematics to enforce constraints, describing its property to maintain target pose as much as possible - original motion trajectory is warped according to the differences in leg lengths - framework also incorporates range of capabilities including: locomotion, object manipulation, gazing, head movements, speech synthesis, and lip-syncing - limited to humanoid or near-humanoid characters

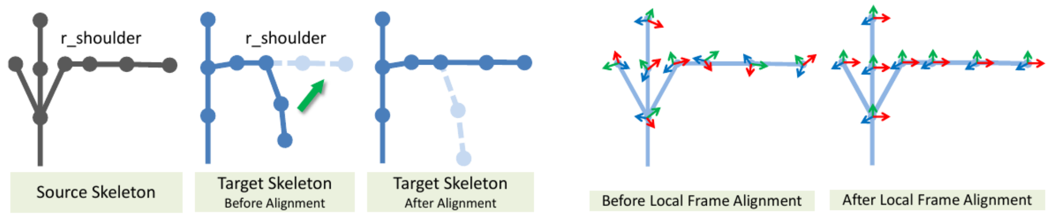

FIGURE 2.12: Image taken from [18]

Ming-Kai Hsieh et al. [19] tackle Retargeting across different articulated figures (e.g., humans, dogs, birds), focusing on transferring and concatenating motions, and generating smooth transitions
- correspondence between the bone structures of the source and target skeletons manually defined, describing automatic skeleton matching techniques as challenging - thus focusing on having a user interface for other aspects of their retargeting process - for joints between source and target mapping one-to-one, many-to-one, or no mapping can be defined
- initial poses are aligned by recursively computing the rotation angles between corresponding bones, from the root to the leaf.
- motion data transfer via converting rotation angles from local to global coordinates and then back to the local coordinates
- for interpolation between two source motions with different skeletal structure a meta-skeleton, combining both skeletons' bone structures, is constructed

### 2.4.3 Numerical approaches

TODOm 2.4.1: was subsection Jacobian based

TODOm 2.4.2: sec beore naive?

expl

- Numerical motion retargeting methods work by solving for the motion of the entire armature simultaneously. This is a departure from traditional techniques that might adjust the position or orientation of limbs independently. This is done by using numerical optimization methods, beeing able to take various aspects of the armature and motion data into consideration.

TODOm 2.4.3: category?

Gleicher [20] describes the challenge to solve Motion Retargeting mathematically because of how to define the quality of a motion, motivating a pragmatic approach.

- propose a numerical solver which utilizes the spacetime constraint approach for skeletons with identical structure but different segment lengths

- require basic features of motion identified as constraints

- spacetime constraints consider the entire motion simultaneously, not just individual frames. minimizes magnitude of the changes and restrict their frequency content To preserve the qualities of the transferred motion - constraints enforced Newton's laws, and the objective function minimized energy consumption

- explain that IK solvers considers each frame independently, adding undesirable high frequencies to a primarily smooth motion - key of preserving high frequencies also feature of motion warpings popularity

- The system works by calculating a motion displacement curve, which represents the difference between an initial estimate of the motion and the final retargeted motion. This displacement is then used to generate the retargeted motion by adding it to the initial estimate. Utilizing motion-displacement maps as data representation and cubic B-splines to avoid encoding high frequencies into the generated motion.

Figure 2.13 shows an Example of motion displacement for an Animation retargeted to a smaller character, weighting the constraint to reach the box high.

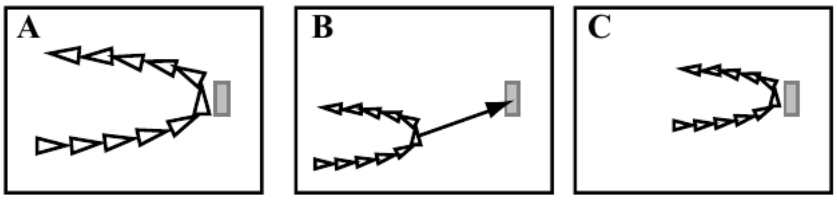

FIGURE 2.13: - A aerial view of a character walking up to, picking up, and carrying away an object. B scaling relative to origin, character does not reach box. C if reaching the box is the only constraint, the entire motion can be translated. Image taken from [20]

- similarities to Jacobian inverse kinematics, utilizing Damped least-squares to minimize constraint residuals across the entire motion
- each step, we construct linear approximation of constraint problem, solved iteratively using a damped pseudo-inverse
- also discuss Motion for Morphing to incorporate scaling of joints during animation
- constraints difficult to represent mathematically as well as hard to define because may not know all the properties required like phyiscal laws, mass of the body and defining which constraints are important to consider to reduce complexity

- challenging to implement, Representation not clearly defined

- Choi and Ko use Inverse Rate Control, which is the Jacobian Inverse Method of Inverse Kinematics, and extend it to be applicaple to tree structures instead of chains without branches. [21] - Their Proposed method, called Online-Motion Retargeting, archieves real-time performance for new characters with different anthropometric proportions. - utilizing jacobian inverse kinematics method fundamentally, Maintaining the characteristic features of the original motion during the retargetting process, avoiding loss of unique motion details.

- The OMR algorithm tracks multiple end-effector trajectories simultaneously from the source character. - Joint angle imitation through minimization of joint angle differences by intelligently using the kinematic redundancies - online: processing data as it comes in, without needing the entire sequence beforehand

- Choi and Ko have also showed a way to imitate joint angles of the source motion by incorporating them as a secondary goal. The primary task tracks given end-effector trajectories and the secondary task is to imitate the joint angle trajectory $\theta$, as best as possible.

- jacobian IK method discussed in 2.2 gives a particular solution - the generalization, called null space, includes all possible solutions by adding a term from the null space:

$$\dot{\theta} = J_1^{\dagger}\dot{x}_1 + (I - J_1^{\dagger}J_1)y_2 \tag{2.3}$$

- where $(I - J_1^{\dagger}J_1)$ projects $y$ onto null space - null space term corresponds to the redundant degrees of freedom

- for primary task $x_1$ and $J_1$ there can be a secondary task $x_2$ $J_2$ added:

$$\dot{\theta} = J_1^{+}\dot{x}_1 + (I - J_1^{\dagger}J_1)J_2^{+}\dot{x}_2 \tag{2.4}$$

- Input trajectories are a continuous input of constraints which applied to the target produce coherent motion.

inverse rate control

- integration of $J_1^{+}\dot{x}_1$ should give $\theta(t)$ - open-loop integration can not eliminate initial tracking error - based on Jacobian pseudoinverse, Closed-loop Inverse Rate Control leads to zero steady state error which means that the error is exponentially convergent to zero for a fixed target position

TODOm 2.4.4: explaination more in detail unnecessary?

- with usage of joint angles $\theta^{src}$ as a secondary target:

$$\dot{\theta}^{des} = J_1^{+}\dot{x}_1 + (I - J_1^{+}J_1)\dot{\theta}^{src}$$

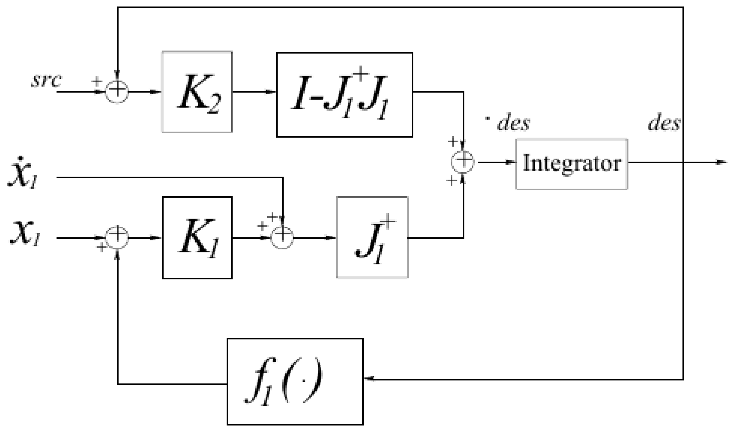

FIGURE 2.14: Image taken from [21]

- Jacobian matrix spans whole armature and contain zeroes where the joint angle and the end-effector have no relation observed different limbs - DOFs for each end-effector also includable, incorporating direction the end-effector has

- algorithm can be used to reduce measurement errors in restoring captured motion by using both measured end-effector data and joint angle data

Monzani et al. [22] introduce an "Intermediate Skeleton" as a bridge between the Performer and End User Skeletons using Inverse rate control to enforce spatial constraints. Developing a plugin for 3D Studio Max with a User Interface.
- Intermediate Skeleton solves this by reorienting its bones - same node structure and local axis system orientations as the End User Skeleton but the bones are oriented to match the Performer Skeleton's bone directions
- describe solution to smooth transitions between captured and corrected postures with multiples constraints, using easing functions to smoothly enable and disable constraints at defined key positions
- user has to indicate a one-to-one correspondence between joints

### 2.4.4 Limb based Retargeting

expl more

- abstract joint chains into limbs to allow for more complex task descriptions - in order to solve limbs indipendently of the whole armature - some tackle problem of automatically defining limbs and correspondences between skeletons

Du Sel et al. [23] tackled real-time motion retargeting for large crowds, developing a plugin for Autodesk Maya.
- use an intermediate skeleton representation called Golaem Skeleton for both source and target skeleton, motion are converted to the intermediate representation for playback on any targets.
- Golaem Skeleton consits of hierarchy of Bone Chains (limbs) defining beginning and end joints, visualized in Figure 2.15

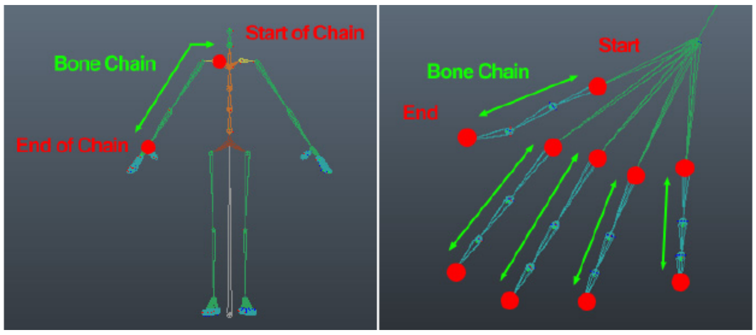

FIGURE 2.15: Example of Golaem Skeleton Bone Chains. Image taken from [23]

- assign different types for Bone chains: pelvis, spine, limb and effector to represent suficient character morphologies
- automate generation of limb nodes, by searching the isomorphic branches and utilize tags generated from chain world position to determine legs or hands
- skeleton motion mapping maps limbs depending on type and tag and allows for surjective, injective and bijective mapping, for example mapping motion of a two-armed biped to a four-armed biped
- defining different movement modes, ability to take or take not restpose differences into consideration to allow for grasping targets or more natural motion playback

- focus on performance to allow for simulation of large crowds, proposing usage of a simplified version of the goalem skeleton, incorporating animation interpolation, mirroring of animation results, utilizing combined FABRIK and analytic IK and pre-computing and interpolating FABRIK results for differt chain lengths called IKCache

- Abdul-Massih et al. [24] retarget motion with focus on maintaining motion style to even non-humanoid characters.
- authors introduce the concept of "Groups of Body Parts" (GBPs), user-defined groupings of joints and motion features to carrying motion style - GBPs are defined by multiple joints with a "base"- and "leading" joint, equivilant to root and endeffector of a joint chain, but GBPs do not have to be a chain
- motion features are seperated into positional and angular features
- positional features represent a path of representative vectors of each GBP - the representative vector points from base to leading joint, capturing basic aspects of the motion, later used in positional constraints in an Inverse Kinematics solver - representative vector can also be scaled depending on the ratio of itself with the matched source GBP vector
- angular amplitude features describe contraction or extension of body parts relative to the neutral input motion, capturing motion style - their angular aplitude constraints controls the range of rotation
- Matched GBPs are then aligned semi-automatically based on their representative vectors direction and starting position - motion transfer is computed on a per GBP basis, beeing able to be used injectively or surjectively, simillar to Du Sel et al.'s Bone Chains. Utilizing a space time approach enforcing constraints
- GBPs definitions for both the source and target are user defined via an interface
- extracted features are convertet into constraints for a full body per frame optimization using space-time optimization

### 2.4.5 Machine Learning Approaches

- due to the complexity of the motion retargeting Problem, machine learning approches have become a popular in recent years providing high quality results - Traditional motion retargeting often requires hand-tuning, goal is to automate as much as possible
- Villegas et al. [25] introduce a novel neural network architecture to retarget motion different skeletal bone lengths and proportions without needing paired motion data.
- uses a recurrent neural networks (RNNs) with differentiable Forward Kinematics (FK) layer as its core component and cycle-consistent adversarial training to achieve unsupervised motion retargetting - network aims to solve the IK problem indirectly with cycle-consistent adversarial training, processing motion sequences on-the-fly, taking advantage of the temporal coherency in motion - for training Motion is retargeted also back to the source character with round trip penalty on difference and discriminator which evaluates its realism - but network limited to same skeleton topology

does mixamo dataset have same skel hierarchy?

Aberman et al. [26] tackle skeletons that differ in number of joints but have topologically equivalent graphs - Deep Motion Representation considers dynamic (joint rotations) and static (joint offsets) components - using skeletal pooling, different skeletons are reduced to a common primal skeleton by a sequence of edge merging operations on degree two joints - the resulting skeleton is also called latent

space, visualized in Figure 2.16, on which motion retargeting is learned and performed
- Retargeting is performed on the dynamic part of the latent space, encoder transforms motions into the shared latent space, while a decoder reconstructs motion on other skeleton, also called skeletal unpooling

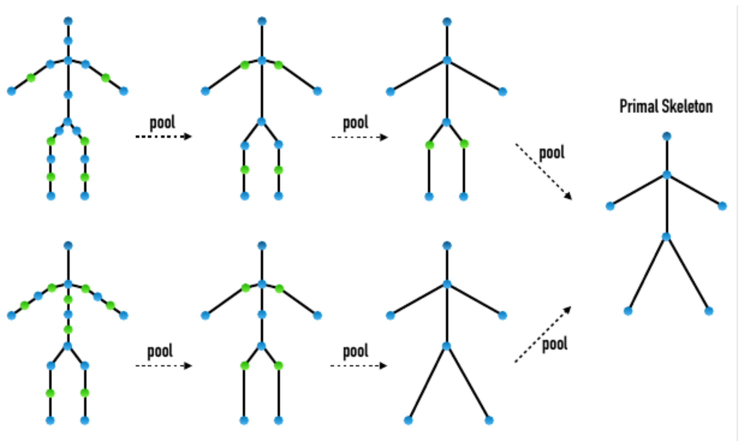

FIGURE 2.16: Image taken from [26]

- encoders learns on Deep Motion Representation, which decouples dynamic (time dependent, rotational) and static (time idependent, offsets) aspects of motion - convolution kernels consider the skeleton structure in both dynamic and static branches seperately during retargeting - model uses a combination of losses for training, notebly encoded-decoded motion should matches input, motion should retain dynamic features, adversarial loss to check for plausibility of retargeted motion, since no ground truth exists and End-Effectors to have simillar velocity - Inverse Kinematics is used to refine foot contact with ground
- limitiations include Challenges retargeting between skeletons with very different T-poses, and dificulty retargeting complex tasks that were not seen in the training set

expand

- Skinned Motion Retargeting with Residual Perception of Motion Semantics & Geometry
- Unsupervised Motion Retargeting for Human-Robot Imitation
- https://arxiv.org/pdf/2402.05115v1

- HMC: Hierarchical Mesh Coarsening for Skeleton-free Motion Retargeting
- https://arxiv.org/pdf/2303.10941v1

- ==Correspondence-Free Online Human Motion Retargeting==
- https://arxiv.org/pdf/2302.00556v3

- OKR: Joint Keypoint Representation for Unsupervised Cross-Domain Motion Retargeting
- https://arxiv.org/pdf/2106.09679v1

- Skinned Motion Retargeting with Dense Geometric Interaction Perception
- https://arxiv.org/pdf/2410.20986v1

- Self-Supervised Motion Retargeting with Safety Guarantee
- https://arxiv.org/pdf/2103.06447v1

- Flow Guided Transformable Bottleneck Networks for Motion Retargeting
- https://arxiv.org/pdf/2106.07771v1

- MoCaNet: Motion Retargeting in-the-wild via Canonicalization Networks
- https://arxiv.org/pdf/2112.10082v2

- Hierarchical Neural Implicit Pose Network for Animation and Motion Retargeting
- https://arxiv.org/pdf/2112.00958v1

- while machine learning approaches can offer good quality retargeting, there is a lack of interactively changing retargeted motion

- new features often require models to be retrained

### 2.4.6 Other approaches

- there are many other novel approaches that cant be categorized clearly or have a special approach seperating it from previously discussed ones

- Hecker et al. [27] tackle the motion retargeting issue during the animation authoring stage
- in a Semantic Approach, animation definition is decomposed into Specialized (character-relative) and Generalized (character-independent) Space
- because animators have to recreate animations in their proposed system, various tools are introduced to help authorin animation, like converting between Specialized and Generalized space
- further specification of body parts are cateorized into Body capabilities (grasper, mouth, spine, root etc.), narrowed down with Spatial and Extent (FrontMost, BackMost, RightMost, etc.) Queries
- Movement Modes introduced similar to other papers applying motion either on the Identity or Rest Relative, Scaling targets, enforcing ground constraints (picking up objects from floor), Secondary Relative Movement (e.g. reaching mouth with hand) or Lookat

cite others

- proposed system incorporated in an interactive editor simultaneously previewing animation while editing, tweaking specialized and generalized parts to ensure motion quality
- to enforce defined constraints a novel IK Solver, Particle IK, is proposed - Particle IKis a two-phase solver; the first phase solves for the spine pose, and the second phase solves for the limb poses, treating the spine as fixed

particle ik in rel 2 IK

- the proposed system was used for animating user-created characters in the game Spore

complete

## 2.5 Automated Rigging

-

### 2.5.1 manual approaches

TODOm 2.5.1: section name

Avril et al. [28] reuse existing rigged characters, transferring their skeleton and skinning weights with distinct mesh topologies, maintaining subtle artistic variations.

- present methods to adjust joint orientation based on vertex weights and retarget skinning weights across different mesh resolutions
- they compute a vertex-to-vertex between source and target meshes, by deforming the source mesh using a set of user defined markers to reduce overhead of defining per vertex correspondences
- joints are placed relative to corresponding vertex joint distances and further refined with Energy minimization techniques while skinning weights are transferred using barycentric coordinates

- streamlines character creation process, allowing refinement of character geometry, skeleton or weights on existing rigged characters - but heavily relies on time-consuming user input to define marker correspondences to archieve good results, requires source and target meshes to be in similar poses and is limited to clean manifold watertight meshes

### 2.5.2 Machine Learning Approaches

- Baran and Popovic [29] introduce a fully automatic rigging method adapting a predefined skeleton to an unrigged character, providing an implementation called Pinocchio

TODOm 2.5.2: is machine learning?

- medial surface is approximated by examining C1-discontinuities in computed signed distance field of the mesh - then spheres are packed based on medial surface position reaching towards the isosurface, placing largest spheres first - essential sphere centers are then connected in a graph on which the predefined skeleton is embedded depending on penalty functions - weights for various hand constructed penalty functions are determined with maximum-margin supervised learning, inspired by support vector machines - finally continuous optimization refines the location of joints and heat-diffusion simulates weights
- pre-existing animations for the predefined skeleton will play more nicely on the rigged character, reducing potential problems of retargeting approaches - drawback character needs to be a closed volume

Xu et al. [30] proposed RigNet, an end-to-end deep learning method to automate character rigging. - Only using 3D character mesh as input, they generate specifically tailored skeletons for a variety of Humanoid and non-humanoid creatures, also automatically computing Surface skin weights
- learns directly from the mesh representation, imitating the joint placement intuition of animators, e.g. spine created closer to the back rather than the medial surface
- present designed a deep modular architecture consisting of Skeletal Joint Prediction, Skeleton connectivity prediction and Skinning prediction.
- Skeletal Joint Prediction, predicts location and number of joints for relevant areas using differentiable clustering scheme "The network learns to displace mesh vertices towards joint locations. A differentiable clustering scheme identifies joint locations

using a neural mesh attention mechanism." - some ambiguity even among animators regarding the number and placement of joints - optional parameter that can control the level-of-detail of the output skeleton (override the learned bandwidth used to determine cluster density,Lowering the band- width results in denser joint placement, while increasing it results in sparser skeletons., zero bandwidth value will cause each displaced vertex to become a joint) - learns to displace mesh geometry towards candidate joint locations - graph neural network extracts topology-and geometry-aware features, a differentiable clustering scheme is then used to extract joints - (incorporates global and local information) - Symmetrization according to the global bilateral symmetry plane before performing clustering improved results

- Skeleton Connectivity Prediction, learns which joints pairs from skeletal joint prediction step should connect to bones using learned shape- and skeleton representation, giving probabilities for each pair, Minimum Spanning Tree algorithm is then used to form a tree starting most likely bones

network learns which joint pairs to connect with bones, forming a hierarchical skeleton tree structure, uses both shape features and joint positions "BoneNet: Predicts the probability of connection between all pairs of joints. A Minimum Spanning Tree (MST) algorithm then generates the final skeleton tree." better - predict a hierarchical tree structure - connecting the joints. The output bone structure is a function of joints predicted from the first stage and shape features - utilizing Euclidean distance and proportion are useful indicators of joint connectivity - considers root as special case having extra network simillar to bone prediction to determine good root bone

- Skinning Prediction - graph neural network predicts skin weights based on intrinsic distances from the mesh vertices to the bones better - also based on a graph neural network operating on shape features and intrinsic distances from mesh vertices to the predicted bones better - use of volumetric geodesic distances from vertices to bones better - using mesh representation capturing spacial relationship of mesh vertices to skeletal bones with volumetric geodecic distances (mesh interior) - inverse of distances used to determine bone priorities, choosing closest bones first - another module is used to compute final skinning weights with learned parameters and a softmax function to get normalized weights

- use a cobination of manually authored loss functions during training simillar to Baran and Popovic - Unlike "Pinocchio", RigNet doesn't rely on pre-defined skeletal templates - direct mesh input avoids pre-processing or lossy conversions - like voxelization Xu et al., 2019 - full solution for both skeleton prediction and skinning, rather than a partial - better captures the anatomical intuition compared to geometric methods.

TODOm 2.5.3: if ref old paper

- not completely invariant to mesh resolution and connectivity - Limited representation of small parts like fingers - control over the level-of-detail of generated skeleton using single bandwith parameter, limited control could be expanded to more parameters

highlight in rignet integration neccessety to cleanup mesh

### 2.5.3 4D approaches

### 2.5.4 Thinning Approaches

list more papers

### 2.5.5 Re-Meshing

# Chapter 3

# Motion Retarget Editor

methodisches vorgehen hier

## 3.1 Processing pipeline

TODOm 3.1.1: goals from related work?

- current research focuses on machine learning - despite ik / limb based methods existing for a long time, there exist no standalone free open source tools or plugins for blender

- Also having an open source foundation opens up community improvements and helps CrossForge mature by protyping features and incorporating them if deemed useful - for a fully automated pipeline, there is a need to keep various parts interactive for interactive testing to verify correct implementation of algorithms

- Noteably, there is a lack of Open Source Implementations of more complex Motion retargeting algorithms and especially frameworks in order to compare and improve motion retargeting.

- Furthermore the process of creating a usable virtual human for various applications remains tedious - goal creating for creating an autonomous virtual human

user interface section?

### 3.1.1 CrossForge

CrossForge [31], developed by Tom Uhlmann at Chemnitz University of Technology, is a A C/C++ Cross-Platform 3D Visualization Framework using OpenGL. - design allows you to use the available CrossForge modules, modify them, or completely replace them with you own OpenGL based implementation and GLSL Shaders. - This flat design, simplicity and direct approach, CrossForge is well suited for educational purposes and computer graphics research.

formulation

- CrossForge allowes for quick implementation of various visualization approaches without beeing restricted the integrated SceneGraph, allowing for quick Prototyping

- while CrossForge already has LinearBlend Skinning and an simple Animation Controller Implemented, it is lacking in many Features, notebly a User Interface for Keyframe Control, Joint Visualization, a Picking System, which had yet to be implemented and will be discussed in the following sections

- because CrossForge is a relatively small Framework compared to Unity or Unreal Engine, many tools like Scene management, User Interfaces or Picking had yet to be implemented - CrossForge lacks User Interface for dynamic loading and unloading of Actors

## 3.2 Classes and Scene Management

smart pointer pos

- reoccuring pattern weak pointer to smart pointer in order to seperate logic to corresponding classes more easily and reduce overhead - weak pointer make it easy to invalidate reference, corresponding class which wants to process another object first has to lock it, checking for its validity, when the referenced object is accessed but invalid by beeing deleted, the corresponding module can act accordingly - reduces state management sagnificantly

### 3.2.1 Character Entity

- The Character Entity Class represents a Single Virtual Character and manages various States related to it.

LISTING 3.1: My Code Example

```
struct CharEntity {
    bool isStatic = false;
    std::unique_ptr<IKSkeletalActor> actor;
    std::unique_ptr<StaticActor> actorStatic;

    // animation
    std::unique_ptr<IKController> controller;
    Animation* pAnimCurr;

    // common
    std::string name;
    T3DMesh<float> mesh;
    SGNGeometry sgn;

    // Picking binding functions
    ...

    // various tool functions
    ...
};
```

- `IKSkeletalActor` and `StaticActor` are renderable objects created from the intermediate Format `T3DMesh` - in order to adapt the underlying mesh data contained in `T3DMesh`

- when a Model is created using an importer, a corresponding CharEntity is created and the geometry node is added to the Scenegraph - in order to differentiate between already rigged Characters and Characters without a Skeleton, unique pointers are used, which can be checked easily for initalization - during initialization T3DMesh is loaded by the actor in order to be visualized using OpenGL - during runtime we need to apply animation changes and mesh operations on the `T3DMesh`

- re-initlizing the renderable actor with the updated `T3DMesh` checks again which features `T3DMesh` has and chooses corresponding actor accordingly

- in order to identify a Char, their name is derived from the filename of the imported asset, if an asset has already the same name, a number will be added

### 3.2.2 Main Scene

- want to dynamically load various models to test quickly without having to restart or delete existing CharEntities during runtime

- since Motion Retargeting requires definition which characters, need to define them
- done using intuitive selection, last click will always define primary charEntity, secondary charEntity is assigned upon selecting a new charEntity that is different from the current one
- operations which require two charEntities can check if both are present beforehand to avoid errors

LISTING 3.2: My Code Example

```
class MotionRetargetScene : public ExampleSceneBase {
    ...
    void mainLoop() override;
    ...
    void renderUI();

    void loadCharPrim(std::string path, IOmeth ioM);
    void storeCharPrim(std::string path, IOmeth ioM);

    struct settings {
        ...
    } m_settings;

    std::vector<std::shared_ptr<CharEntity>> m_charEntities;
    std::weak_ptr<CharEntity> m_charEntityPrim; // currently
        ↪ selected char entity
    std::weak_ptr<CharEntity> m_charEntitySec; // secondary char
        ↪ entity for operations

    SGNTransformation m_sgnRoot;

    ...
};//MotionRetargetScene
```

- in order to inspect model from various angles for comparison, the camera has been extended to rotate to world space axises and toggling between orthographic and perspective projection, controlled with the numpad simmilar to blender

### 3.2.3 Config

- No Configuration System to store Settings in Files for persistent usage accross sessions
- ability to store settings with string or overload with custom type
- to reduce overhead, an extra compile unit is used to define all serialization interfaces in order to seperate Config functionality from the corresponding core modules

- used to store camera position and other settings like paths or load behavior

## 3.3 User Interface

TODOm 3.3.1: Zenodo, to get DOI of github repo? upload others?

For the User Interface ImGui [32] is used. It provides a:
- large and flexible set of Widgets
- very easy integration
- many plugins written for it

- immediate mode, imgui is redrawn every frame
- no separation of states, clean code - imgui docking used to more cleany place ui elements

### 3.3.1 Picking

- in order to interact with objects in the 3D scene, a picking system is needed

- picking objects implemented using ray shooting, transforming mouse click position from screen space back to world space and checking for intersections
- CrossForge already implemented `BoundingVolume` type including AABB and Sphere, checking intersections firstly before checking for intersection with mesh data

cite libigl

- in order to quickly check for intersection libigl is used for important parts like joints, needing conversion from `T3DMesh` to a list of Vectors in a Matrix

- Picker is a class which evaluates a Set of IPickable objects and stores the last and previous clicked Object as a weak pointer reference.

LISTING 3.3: My Code Example

```
class IPickable {
    public:
    virtual void pckSelect() {};
    virtual void pckDeselect() {};
    virtual void pckMove(const Matrix4f& trans) = 0;

    virtual Matrix4f pckTransGuizmo() = 0; // used for guizmo
        ↪ update
    virtual Matrix4f pckTransPickin() = 0; // used for picking
        ↪ evaluation
    virtual const BoundingVolume& pckBV() = 0;
    virtual EigenMesh* pckEigenMesh() { return nullptr; };
};
```

- any class can derive from this interface making it easier adding new types of pickable objects

LISTING 3.4: My Code Example

```
class Picker {
    void pick(std::vector<std::weak_ptr<IPickable>> objects);
    void forcePick(std::weak_ptr<IPickable> pick);
    void start();
    void resolve();
    void reset();
    void update(Matrix4f trans);
    std::weak_ptr<IPickable> getLastPick() {
        return m_pLastPick;
    };
    std::weak_ptr<IPickable> getCurrPick() {
        return m_pCurrPick;
    };
    Matrix4f m_guizmoMat = Matrix4f::Identity();

    void rayCast(Vector3f* ro, Vector3f* rd);
    std::weak_ptr<IPickable> m_pLastPick; // picked object
    std::weak_ptr<IPickable> m_pCurrPick; // last clicked object
    std::weak_ptr<IPickable> m_pPick; // last clicked object
};
```

- with the `Picker` class the application can store references to various types picked objects, most relevant methods are visualized in 3.4.

matrix seperation

- Matrix seperation

### 3.3.2 Scene Control

In Order to apply transformations to picked objects, a gizmo is needed. The term "gizmo" is typically used to refer to a small device or gadget that has been designed for a specific purpose. It often signifies a tool that is capable of performing a particular task in an innovative or efficient manner. The term is informal and can apply to various types of devices.

In the context of graphics programming, gizmos facilitate the manipulation of objects within 3D space. They are widely used in graphics editors to visually represent and control object transformations, most commonly position, rotation, and scale. However, they also cover various other types, such as camera manipulation or mesh editing. They provide intuitive controls that enhance user interaction with the 3D space.

ext

ImGuizmo [33] is a easy to integrate Gizmo Plugin for ImGui.
- ImGuizmo works by taking a reference to an array of floating point number, to stay independent from linear algebra libraries
- a

- LineBox `CForge::LineBox` used to visualize picked object with different highlight strenths in wireframe

### 3.3.3 Widgets

- Outliner
- visualizes Skeletal Hierarchy with collapsable tree nodes containing joint names
- also lists targets
- visibility options only for character settable
- clicking on element in list fetches the corresponding object, when pickable, the picker state gets forced on that object, enabling feedback with highlight visualization

- Animation tab - select animation for playback, set animation speed etc

- Animation tab

- Popup - integrated ImGui Popup had artifacts due to

TODOm 3.3.2: term only render on update?

- Grid
- helps with orentation in 3d space,
- main axis intuitively marked with corresponding color red x, green y, blue z in order to avoid confusion during usage

TODOm 3.3.3: explain ui bindings / implementation in followin parts on the side?

- TODO Matrix seperation

- Popup
- used for

extend

## 3.4 Animation System

CrossForge already provided an implementation for skeletal animation playback using Linear-Blend-Skinning.

ref assimp

### 3.4.1 CrossForge format

For this feature CrossForge implements a direct approach. Assimp, the C++ library used for importing and exporting to various 3D formats. Provides the Inverse Bind Pose matrix. The purpose of this matrix is to transform the joint from global to local space so that local transformation of that joint are applied localy to the weighted vertices when doing linear blend skinning.

### 3.4.2 Sequencer

- A sequencer is a powerful tool in game engines and animation software used for creating and editing cinematic sequences, for

### 3.4.3 Joint Interaction

- todo explain requirement of seperating matrix for gizmo in order or global space transformations to be visualized correctly

- visualizing joints
- using good joint ...

- picking interaction

### 3.4.4 Editing Tools

In order to
- construct restpose

make algorithmic and shorten

LISTING 3.5: My Code Example

```
void IKController::initRestpose() {
    std::function<void(SkeletalAnimationController::SkeletalJoint*
        ↪ pJoint, Matrix4f offP)> initJoint;
    initJoint = [&](SkeletalAnimationController::SkeletalJoint*
        ↪ pJoint, Matrix4f offP) {
        Matrix4f iom = pJoint->OffsetMatrix.inverse();
        Matrix4f t = offP.inverse() * iom;

        // https://math.stackexchange.com/questions/237369/given-
            ↪ this-transformation-matrix-how-do-i-decompose-it-into
            ↪ -translation-rotati
        pJoint->LocalPosition = t.block<3,1>(0,3);
        pJoint->LocalScale = Vector3f(t.block<3,1>(0,0).norm(),
        t.block<3,1>(0,1).norm(),
        t.block<3,1>(0,2).norm());
        Matrix3f rotScale;
        rotScale.row(0) = pJoint->LocalScale;
        rotScale.row(1) = pJoint->LocalScale;
        rotScale.row(2) = pJoint->LocalScale;
        pJoint->LocalRotation = Quaternionf(t.block<3,3>(0,0).
            ↪ cwiseQuotient(rotScale));
        pJoint->LocalRotation.normalize();

        for (uint32_t i = 0; i < pJoint->Children.size(); ++i)
        initJoint(getBone(pJoint->Children[i]),iom);
    };
    initJoint(m_pRoot,Matrix4f::Identity());
}
```

- update restpose
- using cpu vertex skinning to quickly adapt a restpose
- TODO prone to some minor errors due to linear blend skinning deformation on mesh - same issues that happen with linear blend skinning, unrealistic deformations applied to new restpose, and thus visible on any other poses

make algorithmic and shorten

LISTING 3.6: My Code Example

```
void CharEntity::updateRestpose(SGNTransformation* sgnRoot) {
    if(!actor)
    return;

    // apply current pose to mesh data
    for (uint32_t i=0;i<mesh.vertexCount();++i) {
        mesh.vertex(i) = actor->transformVertex(i);
    }

    // forwardKinematics to get updated global pos and rot in
       ↪ m_IKJoints
    controller->forwardKinematics(controller->getRoot());

    for (uint32_t i=0;i<mesh.boneCount();++i) {
        auto* b = mesh.getBone(i);

        //TODOff(skade) bad, assumes mesh idx == controller idx
        IKJoint ikj = controller->m_IKJoints[controller->getBone(i)
            ↪ ];

        // get current global position and rotation
        Vector3f pos = ikj.posGlobal;
        Quaternionf rot = ikj.rotGlobal;

        Matrix4f bindPose = Matrix4f::Identity();
        bindPose.block<3,1>(0,3) = pos;
        bindPose.block<3,3>(0,0) = rot.toRotationMatrix();
        b->InvBindPoseMatrix = bindPose.inverse();
    }

    //TODOf(skade) update animations

    init(sgnRoot);
}
```

- apply transform to Mesh

## 3.5 Inverse Kinematics Implementation

While various Inverse Kinematics Implementations exist, they are usually implemented across various Programming Languages or use different 3D Engines, resulting in vastly different and complex APIs.

To reduce complications, various inverse kinematics algorithms proposed in section 2.2 are re-implemented using CrossForges Animation Controller interface.
- IK play in important Role for implementing limb based methods various - IK methods presented in 2.2 while trying to archieve a common task, show significant differences during motion and thus pose when a target is reached

- Interface for IKSolver `IIKSolver` defines termination settings implementations of solvers need to provide - furthermore additional settings can be added and checked in the Userinterface using `std::dynamic_pointer_cast`

LISTING 3.7: My Code Example

```
class IIKSolver {
    public:
    virtual void solve(std::string segmentName, IKController*
       ↪ pController) {};

    protected:
    float m_thresholdDist = 1e-6f;
    float m_thresholdPosChange = 1e-6f;

    int32_t m_MaxIterations = 50;
};//IIKSolver
```

- subtypes
- `IKController` inherits `SkeletalAnimationController` already defined in CrossForge to reduce overhead and code duplication, only replacing function that need changes
-to stay compatible with `SkeletalAnimationController` functions, types that could need improved structuring or functionality can be extended by using a `std::map<Type*,Extension>` to comfortably extend
- used to assign `SkeletalJoint` not only a local but global component to avoid having to recompute each time on usage

### 3.5.1 Jacobian Method

- Various Sources for Jacobian Inverse Kinematics lack in detail on what specific entries of each cell mean.

- this is due to what the input means

-

### 3.5.2 Heuristic Methods

- explain combined as there shouldnt be as much content?
 - subsectionCCD - contains m_type accessable through dynamic pointer cast for ui

LISTING 3.8: My Code Example

```
    enum Type {
        BACKWARD,
        FORWARD,
    } m_type = BACKWARD;
```

- subsectionFABRIK - contains `std::vector< Vector3f > fbrkPoints` for visualization in framework

requirements for good IK

multiple endeffectors

survey table comparison

## 3.6 Constraints Implementation

section Combined Constraint System (TODO eigenanteil in extra chapter)

### 3.6.1 Target Weighting

(might be mentioned, check)

- while not mentioned by Aristidou et al. [12], Target priorities can be archived by lineary interpolating centoids between optimal sub-base position depending on their Weight.

- in cases where certain position in space cant be reached, chain can optionally be propagated to parent chain and weighted accordingly to avoid problem of stretched limbs

- problem spine is defined as limb, when ideally evaluate?
- evaluate at beginning? (from root to eef / spine first then limbs) - cannot account for centoid / multichain weighting
- evaluate at end? (from eef to root, limbs first then spine) - do not yet know where chain begins, spine evaluated afterwards causes endeffectors to potentially move away from previously reached targets

or the other way around idk yet

- need simillar to fabrik 2 iterations, root to endeffectors, then endeffectors to root globally for the armature to get optimal position
- ik armature evaluated beginning from endeffectors until centoid

### 3.6.2 Angle Constraint Example

- implementing various types of constraints time consuming and challenging - for this example simple Ball- Socket Constraint implemented - Interface for constraint, each inverse kinematics solver can choose a corresponding implementation, best fit for each solver

yet to implement

- cone and sphere

todoff describe visualizer

- rendered via forward pass (primitive factory)

## 3.7 Skeleton Matching

- simplifying problem of matching joints to matching limbs

- during import - can use ICP for algining characters using limb chain positions - prone to error, manual alignmend as option better

- motion retareting initialization sets targets of all chains of primary CharEntity to the selected chains of the secondary CharEntity
- retargeting happens implicitly, during animation playback of the seconday CharEntity, its target points are updated
- because targets are defined in local space, world space transformation is not taken into consideration, in order to view both models clearly, world space transformation can be applied beforehand
- simillar to other editors when manipulating mesh structure, a mode to view objects in local space for aligning is added called edit mode (note that mesh editing is not implemented)

- while testing the new motion retargeting implementation, limbs were matched manually with a popup user interface - could define matching by limb names, but want to autogenerate armature

- only inital guess, user will be able to check matched joints - TODO visualize joint chains with JointPickable

## 3.8 Motion Retargeting

position, general todo stuff

Skeleton-AwareNetworksforDeepMotionRetargeting [26] fig:deep-more1 2.16, shows similarities with limb based approaches

end position, general todo

- Discussed Motion Retargeting Methods Section 2.4 vary significantly in adaptivity

formulation

- Motion Retargeting results still subjective depending on goal of application task

subsection Combined Retargeting Methodologies (TODO eigenanteil in extra chapter)

- because ik methods varry in resluts - propose novel limb based method, which is able to utilize different IK methods for different limbs

### 3.8.1 Armature

- `IKChain` defines a non-branching chain of joints to which a IK Solver is applied to
- Armature consits of a set of joint chains, which are able to intersect, solving an armature solves each individual chain including constraints

- todo order of evaluation of chains problem
- auto create armature

make algorithmic and shorten

LISTING 3.9: My Code Example

```
void CharEntity::autoCreateArmature() {
    if (auto ctrl = controller.get()) {

    std::map<std::string,std::vector<SkeletalAnimationController::
      ↪ SkeletalJoint*>> ikc;

    std::function<void(SkeletalAnimationController::SkeletalJoint*
      ↪ pJoint, std::string name)> propagate;
    propagate = [&](SkeletalAnimationController::SkeletalJoint*
       ↪ pJoint, std::string name) {
        // end current chain and add all childs as new chains
        int cc = pJoint->Children.size();
        // add joint to chain
        if (name != "")
        ikc[name].insert(ikc[name].begin(),pJoint);
        if (cc > 1) {
        for (uint32_t i = 0; i < cc; ++i) {
            auto c =ctrl->getBone(pJoint->Children[i]);
            propagate(c,c->Name);
        }
        } else if (cc == 1) { // add to current chain
            auto c =ctrl->getBone(pJoint->Children[0]);
            propagate(c,name);
        } //else //(cc == 0)  // end effector nothing to do
    };
    propagate(ctrl->getRoot(),"");

    ctrl->m_ikArmature.m_jointChains.clear();
    for (auto& [k,v] : ikc) {
        IKChain nc;
        nc.name = k;
        nc.joints = v;
        ctrl->m_ikArmature.m_jointChains.emplace_back(std::move(nc)
           ↪ );
    }
    }
}
```

### 3.8.2 Joint angle Imitation

- Joint angle Imitation
- many papers dont explain how how to transfer motion of

- despite having similar rest pose position, non-redundant parts of skinning matrix still have influence on how motion data is applied to a rig - thus cant simply apply motion data from one rig to another

- recreate restpose, not always feasable, as some limbs may not exist in the other rig, or restpose adaptation causes unpleasing deformations
- we expect that simply applying motion data causes problems described in Section 2.4.1
- still imitating Joints still provides good initial guesses, in which the task of IK is to cleanup artifacts

make algorithmic and shorten + expl

LISTING 3.10: My Code Example

```
    std::function<void(SkeletalAnimationController::SkeletalJoint*
       ↪ j, Matrix4f parentT)> imitate;

    imitate = [&](SkeletalAnimationController::SkeletalJoint* jt,
       ↪ Matrix4f parentT) {
    IKChain* ct = nullptr;
    if (jointToChain[jt].size() > 0)
    ct = jointToChain[jt][0]; //TODOff(skade) multiple chains?
    bool noMatch = true;
    if (ct) {
        //TODO(skade) find corresponding retarget chain
        int is = 0;
        for (int it = 0; it < m_ikcorr.size();++it)
        if (&tCtrl->m_ikArmature.m_jointChains[it] == ct)
        is = m_ikcorr[it];
        IKChain& cs = sCtrl->m_ikArmature.m_jointChains[is];

        int i = std::distance(ct->joints.begin(),std::find(ct->
            ↪ joints.begin(),ct->joints.end(),jt));
        int matchIdx = jointIndexingFunc(i,cs,*ct);

        if (matchIdx != -1) {
            SkeletalAnimationController::SkeletalJoint* js = cs.
                ↪ joints[matchIdx];
            Eigen::Matrix4f jsT = CForgeMath::translationMatrix(js
                ↪ ->LocalPosition)
            * CForgeMath::rotationMatrix(js->LocalRotation)
            * CForgeMath::scaleMatrix(js->LocalScale);

            Eigen::Matrix4f parentS = Matrix4f::Identity();
            {
                auto* jsc = js;
                Matrix4f adjS = Matrix4f::Identity();
                while (jsc->Parent != -1) {
                    jsc = sCtrl->getBone(jsc->Parent);
                    Eigen::Matrix4f jscT = CForgeMath::
                        ↪ translationMatrix(jsc->LocalPosition)
                    * CForgeMath::rotationMatrix(jsc->LocalRotation
                        ↪ )
                    * CForgeMath::scaleMatrix(jsc->LocalScale);
                    parentS = jscT * parentS;

                    ////TODO(skade) adj
                    //// with adjustment
                    //parentS = jscT * adjS * parentS;
                    //Matrix4f adjS = js->OffsetMatrix.inverse() *
                        ↪ adjS;
                }
            }

            //Matrix4f t = jt->OffsetMatrix * parentT.inverse() *
                ↪ parentS * js->OffsetMatrix.inverse() * jsT;
            //Matrix4f t = jt->OffsetMatrix * parentT.inverse() *
                ↪ parentS * js->OffsetMatrix.inverse() * jsT;

            // new local transform
            Matrix4f t = Matrix4f::Identity();

            // parent target
            if (jt->Parent != -1 && js->Parent != -1) {
                auto* jtp = tCtrl->getBone(jt->Parent);
                auto* jsp = sCtrl->getBone(js->Parent);

                // current global transform of retargeted parent
```

```
                    //Matrix4f parentGlobal =  parentT * jtp->
                        ↪ OffsetMatrix;
                    //Matrix4f parentGlobalS = parentS * jsp->
                        ↪ OffsetMatrix;

                    // allign next transform so global transform of
                        ↪ target and source are identical
                    //t = jsT;

                    ////TODO(skade) adj
                    //// get parent relative joint change of restpose
                    //Matrix4f adjS = js->OffsetMatrix.inverse() * jsp
                        ↪ ->OffsetMatrix;
                    //Matrix4f adjT = jt->OffsetMatrix.inverse() * jtp
                        ↪ ->OffsetMatrix;
                    //adjS.block<3,1>(0,3) = Vector3f::Zero();
                    //adjT.block<3,1>(0,3) = Vector3f::Zero();
                    //t = adjT.inverse() * parentT.inverse() * parentS
                        ↪ * adjS
                    //  * jsT * js->OffsetMatrix * jt->OffsetMatrix.
                        ↪ inverse();

                    // correct but doesnt account for rest pose
                        ↪ differences
                    //t = parentT.inverse() * js->SkinningMatrix * jt->
                        ↪ OffsetMatrix.inverse();
                    //t = parentT.inverse() * parentS * jsT * js->
                        ↪ OffsetMatrix * jt->OffsetMatrix.inverse();

                    t = parentT.inverse() * parentS
                    * jsT * js->OffsetMatrix * jt->OffsetMatrix.inverse
                        ↪ ();

                    //parentS = parentS * js->OffsetMatrix * jsT;

                    // correct
                    parentT = parentT * t;
                    // but also means:
                    //parentT = parentS
                    //  * jsT * js->OffsetMatrix * jt->OffsetMatrix.
                        ↪ inverse();

                    ////TODO(skade) adj
                    //parentT = parentS * adjS
                    //  * jsT * js->OffsetMatrix * jt->OffsetMatrix.
                        ↪ inverse();
                }

                { // set local rotation of joint
                    Vector3f p,s; Quaternionf r;
                    MRMutil::deconstructMatrix(t,&p,&r,&s);
                    //jt->LocalPosition = p;
                    jt->LocalRotation = r;
                    //jt->LocalScale = s;
                }
                noMatch = false;
            }
```

```
    }
    if (noMatch) {
        Eigen::Matrix4f jtT = CForgeMath::translationMatrix(jt->
            ↪ LocalPosition)
        * CForgeMath::rotationMatrix(jt->LocalRotation)
        * CForgeMath::scaleMatrix(jt->LocalScale);

        ////TODO(skade) adj
        //Matrix4f adjT = Matrix4f::Identity();
        //if (jt->Parent != -1) {
            //  auto* jtp = tCtrl->getBone(jt->Parent);
            //  Matrix4f adjT = jt->OffsetMatrix.inverse() * jtp->
                ↪ OffsetMatrix;
            //}
        //parentT = parentT * jtT * adjT;

        parentT = parentT * jtT;
    }

    for (auto child : jt->Children) {
        imitate(tCtrl->getBone(child), parentT);
    }
};
//TODO(skade) make toggable
imitate(tCtrl->getRoot(), Eigen::Matrix4f::Identity());
```

- root special bone - root handling

impl rotation, make optional

LISTING 3.11: My Code Example

```
    // copy root position
for (int i=0;i< tCtrl->boneCount();++i) {
    auto jt = tCtrl->getBone(i);
    if (jt->Parent == -1) {
        auto jt = tCtrl->getBone(i);
        for (int j=0;j< sCtrl->boneCount();++j) {
            auto js = sCtrl->getBone(j);
            if (js->Parent == -1) {
                jt->LocalPosition = js->LocalPosition;
                break;
            }
        }
        break;
    }
}
```

- for a given limb pair we need to determine which joint pairs which should transfer the motion
- for this an adaptable index function can be used

- bijective ideal case, mapping corresponding joints, disregarding of length to the other joint, will produce wrong endeffector positions in case matched joints differ in length strongly

- in case the mapping is not bijective, we need to find a good indexing function
- injective will cause redundant joints to be alligned with another joint in the longer chain, beeing then used during correction in ik

- surjective more complex, one joint should imitate the the transformation of 2 or more joints, which need to be computed using offset matrix to correctly take restpose orentation into account

### 3.8.3 kinematic chain scaling

- a natural desire for Motion Retargeting would be to support animation transfer between not only vastly different skeletal structures but also sizes and heights

- it is hard to define how a good retargeting should look like, depending on the desired task:

- e.g. retarget animation of virtual character holding a box, with different arm lengths
- should the box keep its position in space? can its size change? if not motion can cause
- same for gait motion, walk speed or root height

- while machine learning approaches find good looking results, they fail in this regard because they often cannot be adapted or extended for specific needs or unforseen requirements

- need user defined adaptivity, in order to satisfy various desired tasks
- compare matched limb lengths and scale target position relative to limb scale root

formula

let $T$ be target position and $R$ the root position of the inspected chain $C$:

$$T_{new} = \frac{|T - R| \cdot |C_{tar}|}{|C_{src}|} + R$$

- depending on task specification, this term can be extended using linear interpolation term $d$:

$$T_{new} = \frac{|T - R| \cdot (d(|C_{tar}| - |C_{src}|) + |C_{src}|)}{|C_{src}|} + R$$

- for $d = 0$ source target position is used, while for $d = 1$ rescaled target position is used
- of course more heuristics can be used in order to keep a certain target height fix

### 3.8.4 Editor Main Loop

section name?

-
- algo that gets applied every frame

edit mode other name

**Algorithm 3** main loop editor

```
1: procedure MAIN LOOP EDITOR, EXECUTED EACH FRAME
2:     while Editor not terminated do
3:         label START
4:         if no input or action flag set then
5:             - wait for input events
6:             goto START
7:         end if
8:         - apply joint imitation (or other Motion Retargeting properties)
9:         - apply IK to reach targets of each limb
10:        for all Character do
11:            if Character has active animation then
12:                apply animation
13:            else
14:                solve armature for ik targets
15:            end if
16:        end for
17:        Edit Mode
18:        if no ImGui element hovered then
19:            handle Object deletion
20:            handle Picking for all pickables
21:            update Camera
22:        else
23:            handle view manipulate widget
24:        end if
                                                        ▷ Render Scene
25:        Shadow Pass
26:        Geometry Pass
27:        Lighting Pass
                                                        ▷ Render GUI
28:        Forward Pass
29:        render visualizers
30:        render ImGui interfaces
31:        Swap Buffers
32:    end while
33: end procedure
```

## class hierarchy overview

scene

TODOm 3.8.1: how ideally split? + position?

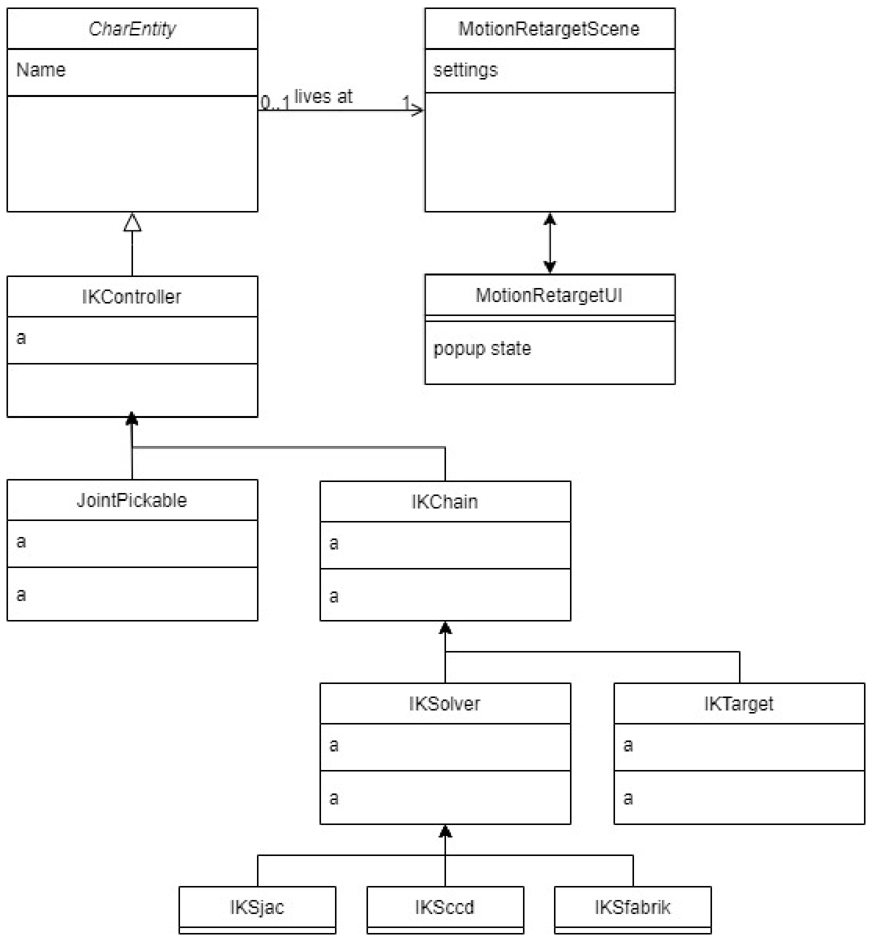

FIGURE 3.1

### 3.8.5 Comparison of IK Methods

- while fabrik author x stated that fabrik produces more natural results this is not necessarily true for all kinds of motions.
For example, while it may be true that fabrik produces more plausible results for grasping or low energy reaching motions.
Motions that require more force or result as a measure of lowering energy consumption like runing or pushing motions. jacobian inverse has the ability to more naturaly correct these due to its way of distributing change and thus work across all bones instead of ones closes to the endeffector

image of comparison without joint angle imitation

### 3.8.6 Comparison with other existing MotionRetargeting Methods

- todo other methods yet to be implemented

- possibility to use error metric to compare quality of
- by comparing joint position and angle or surface properties in case of differering skeletons
- todo

## 3.9 Import and Export

- CrossForge already used Assimp as an interface - Assimp supports various Formats including simple ones like obj,ply,stl,bvh but also more complex ones e.g. fbx, gltf - assimp uses a unified intermediate format which is then used to transfer data between T3DMesh

- during import meshes are auto-scaled to fit into the predefined scaling using the AABB, can be disabled

### 3.9.1 Model Data

- assimp has various issues regarding round trip export and import, not preserving bone structure, namings or graphs in some cases, but more importantly - assimp is not suited for editing meshes, intended workflow of the MotionRetarget Editor would be to export the Character in blender, rig and retarget motion in the proposed editor, and export back to import into blender - assimp cant ensure that the imported mesh preserves structure, as different formats might not support the same representable surface properties of each format, assimp tries to cover as many formats - assimps intermediate format focuses on providing an easy to use integration, beeing close to how Graphics APIs expect it, for example ensuring seams are handled by splitting the mesh up at corresponding positions

assimp is prone to change the underlying models topology slightly in some cases to ensure most compatability across multiple formats, meaning that a round trip, import export without changing anything, produces a different file - which can be undireable when the goal is to produce a clean model

- in order to improve round trip import and export CrossForge had a native gltf interface prototype, but incomplete and various issues

- with gltf 2.0, gltf has become a good fromat rivaling fbx, in comparison to its predecessor version 1.0 pbr materials, binary representation of data with glb or morphtargets are great additions to form a good open source alternative to the propietary industry standard from autodesk - while assimp is a popular tool for importing assets for prototyping engines, its intermediate format and wide support of formats limit the correctness of bidirectional export and import, depending on file format certain kinds of data get changed, for example triangle count or order - because blender focuses 3D creation, its internal format and exporting tools support a wide range of options for export and thus avoiding redundencies - because blender is foss, there is no need to implement various file format interfaces for CrossForge - with crossforges direct gltf export interface, correct export is assured, optionally assimp can be used as well but might have problems exporting correctly

TODOm 3.9.1: better? subsectionModel Data subsectionAnimation Data

- bvh, which most commonly motion capture databases use, is supported by assimp

future

- implement native bvh

### 3.9.2 Armature Data

- import / export armature, limb chain definitions and constraints for Motion Retargeting

## 3.10 Additional Library Integration

- not every existing tool is written in C++ - while bindings other languages exist - in order to integrate other tools there needs - many - Rignet recommends using Anaconda, a environemnt wrapper for python libraries - starting such an environment from an embedded pyhton in C++ is challenging - with each additional tool there would be effort required to find or develop binding

what is std::system exactly

- solution, simply use std::system, creating a subprocess to run a command, either executing an executable or script - in order to transfer data, either command line arguments or preferablyfiles can be

- Interface for binding Auto-Rigging solutions
- AROptions is template type, used to define struct containing options
- ImGui interfeces uses these structs to compactly define a set of options

Listing 3.12: My Code Example

```cpp
template<typename AROptions>
class IAutoRigger {
    virtual void rig(T3DMesh<float>* mesh, AROptions options) = 0;
};
```

title

### 3.10.1 Rignet

- `T3DMesh` format not abstracted, close to visualization, causes vertices to be doubled at texture seams or other parts, furthermore - no detailed instructions from the provided python implementation on how to use correctly

explain bandwidth and threshold

- threshold - bandwidth

- start corresponding conda environment, and using the enclosed quick start script with minor changes to specify model and parameter input, as well as processing folder - option to use previous computed cache

- Rignet expects - vertices need to be merged in order to provide optimal results
- rignet provides the resulted rig in a custom format that needs to be parsed

- because there are no standardized formats for describing a skeleton including skinning weights, Rignet outputs a custom format

explain format parsing

# Chapter 4

# Conclusion and Future Work)

## 4.1 Editor Improvements

- CrossForge problem `T3DMesh` representation not suited for mesh editing - need seamless internal format - extending editor animation tools

## 4.2 Blender Addon

cite blender

- while blender would have been an alternative suspect to implement the pipeline, highly abstract source code and potentially limiting python API as well as debugging possibilities made it unattractive for prototyping - still blender one of the most used animation and modelling editors, many users would benefit from an improved Motion Retargeting solution - rignet plugin for blender already exists - that no proper motion retargeting solution exists for blender highlights challenge of implementation - Addon for popular open source game engine godot also useful

## 4.3 SMPL fitting

explain smpl

- option to fit the learned human model smpl to a 3d scan and then transfer surface properties via projection

CrossForge already has an working test of transferring textures by projecting triangle vertices of a scan to the nearest SMPL vertex using normals, but has yet to be properly implemented in order to be integreted into the propsed editor

Furthermore transferring geometric surface propertices could be done using neighbor evaluation with laplacian difference.

more in depth

- Compared to Rignet, SMPL already has rigged hands and face, despite its limitation to humans it would provide a great alternative

## 4.4 Other Ideas

- Utilizing Skinning Alternatives (direct delta mush, goes into autorigging simplifications) - Clothing (clothing simulation integration)
- Motion Blending, combine motion data of multiple motions during retargeting

was already proposed in other paper ref

in appendix

- TODO MetaHuman (UE5) provides an excellent quality with facial and hand rig - but creation restricted to existing toolset provided by environment - clothing has to be recreated - cant use scan

# Appendix A

# Existing IK Tools

- Jacobian Methods Impl Because of the Mathematical complexity of Jacobian Methods, implementations are hard to find.

- CCD Impl

TODO Tex

- Fabrik Impl

- a - Final IK ik collection for unity - http://www.root-motion.com/finalikdox/htm
- paid - no source code

formulate

formulate

comparison from FABRIK paper

# Bibliography

[1] Thomas Kronfeld. "Themenschwerpunkte Informatik: Virtual Humans 3. Kinematik". 2022.

[2] Sébastien Moya and Floren Colloud. "A FAST GEOMETRICALLY-DRIVEN PRIORITIZED INVERSE KINEMATICS SOLVER". In: ().

[3] A. Aristidou et al. "Inverse Kinematics Techniques in Computer Graphics: A Survey". In: *Computer Graphics Forum* 37.6 (Sept. 2018), pp. 35–58. ISSN: 0167-7055, 1467-8659. DOI: 10.1111/cgf.13310. URL: https://onlinelibrary.wiley.com/doi/10.1111/cgf.13310.

[4] Jeff Lander. "Oh My God, I Inverted Kine! 09/98: Graphic Content". In: (1998).

[5] Thomas Kronfeld. "Themenschwerpunkte Informatik: Virtual Humans 4. Inverse Kinematik". 2022.

[6] Samuel R Buss. "Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares Methods". In: ().

[7] Ben Kenwright. "Inverse Kinematics – Cyclic Coordinate Descent (CCD)". In: *Journal of Graphics Tools* 16.4 (Oct. 2012), pp. 177–217. ISSN: 2165-347X, 2165-3488. DOI: 10.1080/2165347X.2013.823362. URL: http://www.tandfonline.com/doi/abs/10.1080/2165347X.2013.823362.

[8] L.-C.T. Wang and C.C. Chen. "A Combined Optimization Method for Solving the Inverse Kinematics Problems of Mechanical Manipulators". In: *IEEE Transactions on Robotics and Automation* 7.4 (Aug. 1991), pp. 489–499. ISSN: 1042296X. DOI: 10.1109/70.86079. URL: http://ieeexplore.ieee.org/document/86079/.

[9] Andreas Aristidou and Joan Lasenby. "FABRIK: A Fast, Iterative Solver for the Inverse Kinematics Problem". In: *Graphical Models* 73.5 (Sept. 2011), pp. 243–260. ISSN: 15240703. DOI: 10.1016/j.gmod.2011.05.003. URL: https://linkinghub.elsevier.com/retrieve/pii/S1524070311000178.

[10] Masanori Sekiguchi and Naoyuki Takesue. "Fast and Robust Numerical Method for Inverse Kinematics with Prioritized Multiple Targets for Redundant Robots". In: *Advanced Robotics* 34.16 (Aug. 17, 2020), pp. 1068–1078. ISSN: 0169-1864, 1568-5535. DOI: 10.1080/01691864.2020.1780151. URL: https://www.tandfonline.com/doi/full/10.1080/01691864.2020.1780151.

[11] Ronan Boulic et al. "Evaluation of On-Line Analytic and Numeric Inverse Kinematics Approaches Driven by Partial Vision Input". In: *Virtual Reality* 10.1 (May 2006), pp. 48–61. ISSN: 1359-4338, 1434-9957. DOI: 10.1007/s10055-006-0024-8. URL: http://link.springer.com/10.1007/s10055-006-0024-8.

[12] Andreas Aristidou, Yiorgos Chrysanthou, and Joan Lasenby. "Extending FABRIK with Model Constraints". In: *Computer Animation and Virtual Worlds* 27.1 (Jan. 2016), pp. 35–57. ISSN: 1546-4261, 1546-427X. DOI: 10.1002/cav.1630. URL: https://onlinelibrary.wiley.com/doi/10.1002/cav.1630.

[13] Jane Wilhelms and Allen Van Gelder. “Fast and Easy Reach-Cone Joint Limits”. In: *Journal of Graphics Tools* 6.2 (Jan. 2001), pp. 27–41. ISSN: 1086-7651. DOI: 10.1080/10867651.2001.10487539. URL: http://www.tandfonline.com/doi/abs/10.1080/10867651.2001.10487539.

[14] Kris Hauser. *Robotic Systems (Draft)*. University of Illinois at Urbana-Champaign. URL: https://motion.cs.illinois.edu/RoboticSystems/InverseKinematics.html.

[15] Chris Hecker. “My Adventure with Inverse Kinematics”.

[16] Joris De Schutter et al. “Constraint-Based Task Specification and Estimation for Sensor-Based Robot Systems in the Presence of Geometric Uncertainty”. In: *The International Journal of Robotics Research* 26.5 (May 2007), pp. 433–455. ISSN: 0278-3649, 1741-3176. DOI: 10.1177/027836490707809107. URL: https://journals.sagepub.com/doi/10.1177/027836490707809107.

[17] Chen Tang et al. “Motion Retargeting for Characters with Heterogeneous Topologies”. In: *2012 5th International Congress on Image and Signal Processing*. 2012 5th International Congress on Image and Signal Processing (CISP). Chongqing, Sichuan, China: IEEE, Oct. 2012, pp. 756–760. ISBN: 978-1-4673-0964-6 978-1-4673-0965-3 978-1-4673-0963-9. DOI: 10.1109/CISP.2012.6469919. URL: http://ieeexplore.ieee.org/document/6469919/.

[18] Andrew Feng et al. “Automating the Transfer of a Generic Set of Behaviors onto a Virtual Character”. In: *Motion in Games*. Ed. by Marcelo Kallmann and Kostas Bekris. Red. by David Hutchison et al. Vol. 7660. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 134–145. ISBN: 978-3-642-34709-2 978-3-642-34710-8. DOI: 10.1007/978-3-642-34710-8_13. URL: http://link.springer.com/10.1007/978-3-642-34710-8_13.

[19] Ming-Kai Hsieh, Bing-Yu Chen, and Ming Ouhyoung. “Motion Retargetting and Transition in Different Articulated Figures”. In: *Ninth International Conference on Computer Aided Design and Computer Graphics (CAD-CG’05)*. Ninth International Conference on Computer Aided Design and Computer Graphics (CAD-CG’05). Hong Kong, China: IEEE, 2005, pp. 457–462. ISBN: 978-0-7695-2473-3. DOI: 10.1109/CAD-CG.2005.59. URL: http://ieeexplore.ieee.org/document/1604675/.

[20] Michael Gleicher. “Retargetting Motion to New Characters”. In: *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH ’98*. The 25th Annual Conference. Not Known: ACM Press, 1998, pp. 33–42. ISBN: 978-0-89791-999-9. DOI: 10.1145/280814.280820. URL: http://portal.acm.org/citation.cfm?doid=280814.280820.

[21] Kwang-Jin Choi and Hyeong-Seok Ko. “On-Line Motion Retargetting”. In: (1999).

[22] Jean-Sébastien Monzani et al. “Using an Intermediate Skeleton and Inverse Kinematics for Motion Retargeting”. In: *Computer Graphics Forum* 19.3 (Sept. 2000), pp. 11–19. ISSN: 0167-7055, 1467-8659. DOI: 10.1111/1467-8659.00393. URL: https://onlinelibrary.wiley.com/doi/10.1111/1467-8659.00393.

[23] Yann Pinczon Du Sel, Nicolas Chaverou, and Michaël Rouillé. “Motion Retargeting for Crowd Simulation”. In: *Proceedings of the 2015 Symposium on Digital Production*. DigiPro ’15: The Digital Production Symposium. Los Angeles California: ACM, Aug. 8, 2015, pp. 9–14. ISBN: 978-1-4503-3718-2. DOI: 10.

1145/2791261.2791264. URL: https://dl.acm.org/doi/10.1145/2791261.2791264.

[24] M. Abdul-Massih, I. Yoo, and B. Benes. "Motion Style Retargeting to Characters With Different Morphologies". In: *Computer Graphics Forum* 36.6 (Sept. 2017), pp. 86–99. ISSN: 0167-7055, 1467-8659. DOI: 10.1111/cgf.12860. URL: https://onlinelibrary.wiley.com/doi/10.1111/cgf.12860.

[25] Ruben Villegas et al. *Neural Kinematic Networks for Unsupervised Motion Retargetting*. Apr. 16, 2018. arXiv: 1804.05653 [cs]. URL: http://arxiv.org/abs/1804.05653. Pre-published.

[26] Kfir Aberman et al. "Skeleton-Aware Networks for Deep Motion Retargeting". In: *ACM Transactions on Graphics* 39.4 (Aug. 31, 2020). ISSN: 0730-0301, 1557-7368. DOI: 10.1145/3386569.3392462. arXiv: 2005.05732 [cs]. URL: http://arxiv.org/abs/2005.05732.

[27] Chris Hecker et al. "Real-Time Motion Retargeting to Highly Varied User-Created Morphologies". In: (2008).

[28] Quentin Avril et al. "Animation Setup Transfer for 3D Characters". In: *Computer Graphics Forum* 35.2 (May 2016), pp. 115–126. ISSN: 0167-7055, 1467-8659. DOI: 10.1111/cgf.12816. URL: https://onlinelibrary.wiley.com/doi/10.1111/cgf.12816.

[29] Ilya Baran and Jovan Popovic. "Automatic Rigging and Animation of 3D Characters". In: (2007).

[30] Zhan Xu et al. "RigNet: Neural Rigging for Articulated Characters". In: *ACM Transactions on Graphics* 39.4 (Aug. 31, 2020). ISSN: 0730-0301, 1557-7368. DOI: 10.1145/3386569.3392379. URL: https://dl.acm.org/doi/10.1145/3386569.3392379.

[31] Tom Uhlmann. *CrossForge: A Cross-Platform 3D Visualization and Animation Framework for Research and Education in Computer Graphics*. 2020. URL: https://github.com/Tachikoma87/CrossForge.

[32] Omar Cornut. *ImGui*. URL: https://github.com/ocornut/imgui.

[33] Cedric Guillemet. *ImGuizmo*. 2016. URL: https://github.com/CedricGuillemet/ImGuizmo.

TECHNISCHE UNIVERSITÄT CHEMNITZ

**Zentrales Prüfungsamt**
**Selbstständigkeitserklärung**

| | |
|---|---|
| Name:<br><br>Vorname:<br><br>geb. am:<br><br>Matr.-Nr.: | Bitte beachten:<br><br>1. Bitte binden Sie dieses Blatt am Ende Ihrer Arbeit ein. |

Selbstständigkeitserklärung*

Ich erkläre gegenüber der Technischen Universität Chemnitz, dass ich die vorliegende
selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe.

Die vorliegende Arbeit ist frei von Plagiaten. Alle Ausführungen, die wörtlich oder inhaltlich aus anderen Schriften entnommen sind, habe ich als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch nicht als Prüfungsleistung eingereicht und ist auch noch nicht veröffentlicht.

Datum: …………………………………….          Unterschrift: ………………………………………………………………………

* Statement of Authorship

I hereby certify to the Technische Universität Chemnitz that this thesis is all my own work and uses no external material other than that acknowledged in the text.

This work contains no plagiarism and all sentences or passages directly quoted from other people's work or including content derived from such work have been specifically credited to the authors and sources.

This paper has neither been submitted in the same or a similar form to any other examiner nor for the award of any other degree, nor has it previously been published.