eTryOn - Virtual try-ons of garments enabling novel human fashion interactions

Project Title:	**eTryOn - Virtual try-ons of garments enabling novel human fashion interactions**
Contract No:	951908 - eTryOn
Instrument:	Innovation Action
Thematic Priority:	H2020 ICT-55-2020
Start of project:	1 October 2020
Duration:	24 months

Deliverable No: D2.2

First working version of the avatar-garment simulation software

Due date of deliverable:	30 November 2021
Actual submission date:	8 December 2021
Version:	Final
Main Authors:	Metail

Project funded by the European Community under the H2020 Programme for Research and Innovation.

Project ref. number	951908
Project title	eTryOn – Virtual try-ons of garments enabling novel human fashion interactions

Deliverable title	First working version of the avatar-garment simulation software
Deliverable number	D2.2
Deliverable version	Version 1.0
Contractual date of delivery	30 Nov 2021
Actual date of delivery	8 December 2021
Deliverable filename	eTryOn_D2.2.docx
Type of deliverable	Demonstrator
Dissemination level	PU
Number of pages	55
Workpackage	WP2
Task(s)	T2.1-T2.4
Partner responsible	Metail
Author(s)	Dongjoe Shin, David Gavilan, Robert Boland, Jim Downing (Metail)
Editor	Elisavet Chatzilari (CERTH)
Reviewer(s)	Thomas De Wilde (QuantaCorp)

Abstract	In this deliverable the basic working version of the avatar-garment simulation software will be provided in order to be used for iterative testing.
Keywords	3D garments, Vstitcher Browzwear, Simulation algorithms

Copyright

Deliverable history

Version	Date	Reason	Revised by
1.0	08/10/21	Table of Contents	Robert Boland
1.1	29/11/21	Initial version	Robert Boland, David Gavilan, Dongjoe Shin
1.2	05/12/21	Comments by	Thomas De Wilde
1.3	07/12/21	Revised version	Robert Boland
2.0	08/12/21	Final version	Elisavet Chatzilari

List of abbreviations and Acronyms

Abbreviation	Meaning
VS	VStitcher
BW	Browzwear
PBS	Physics Based Simulation
CLI	Command Line Interface
DL	Deep Learning
ML	Machine Learning
CNN	Convolutional Neural Network
GCN	Graph CNN
KNN	K-Nearest Neighbors algorithm
IoU	Intersection over Union
RoI	Region of interest
DB	Database
BMI	Body Mass Index
ND	Neural Dynamics
ABC	Alembic (file format)
FBX	FilmBox (file format)
PCA	Principal Component Analysis
DRAPE	Dressing Any Person
LSTM	Long short-term memory
BERT	Bidirectional Encoder Representations from Transformers
QC	Quality Control

List of Figures

Table of Contents

1. Executive summary

The objective of eTryOn's WP2 is to investigate and develop an automatic, cost-effective and systematic method to fit digitally-designed garments to the 3D avatars generated in WP1. As such the objective consists of both:

- Generating a collection of digital garments
- Automatic fitting of the digital garments in both still images (DressMeUp app) and in real-time (MagicMirror app)

At a high level, the goal can be described as **creating realistic visualisations of 3D garments on a person's body**. These visualisations need to be rendered in software accessible to end users.

The current deliverable builds on the research delivered in D2.1 and uses the garments created in that deliverable to create a first working version of the avatar-garment simulation software.

Section 3 describes in detail the core innovation in the avatar-garment simulation work. We have achieved many targets in this innovation objective and describe that in detail in this document. We explain how we use VStitcher animations as a ground truth to model cloth simulation in Unity-Obi. We identify vertex movement of the garment for a given animation and then use Nevergrad optimisation to work out the optimal Obi parameters for that garment on a given avatar. Combining this with the body tracking described in sections 4 (for Snapchat) and 5 (for Unity) allowed us to create initial working versions of the MagicMirror app. The detail of the difficulties encountered and solutions to these issues is provided in this deliverable.

As well as the MagicMirror (in sections 4 and 5), we also describe the progress of the DressMeUp (section 6) and VR Designer (section 7) applications. In both of these applications we have successfully created an end-to-end prototype. In the DressMeUp app, we have taken the scans from WP1 and integrated them into our pipeline. As part of this process we have identified some of the current limitations around avatar-garment alignment. We have built a framework for evaluating the results and will use this to target improvements over the next few months. In the VR designer app, we explain the decision to directly use VStitcher animations for best results and the next steps to integrate more realistic looking avatars into that application.

2. Introduction

In the previously completed deliverable (D2.1) we described both the process of creating a 3D garment collection and our first research into how to take these assets and produce realistic simulation on an avatar. In this deliverable, we have furthered this research and created the first working version of the avatar-garment simulation software.

In the eTryOn Project we are developing three different applications, all of which provide innovative ways to interact with avatars and virtual garments. Although the interaction environment and requirements are different for each application, they all start from the same base - digital 3D garments which have been designed for production using professional 3D design software (Browzwear's VStitcher in our case). These garments are then transformed into a compatible form for each application. In this document we will describe the shared core research of cloth simulation, and then go into detail of each of the use cases.

At a high level, the research into cloth simulation uses Browzwear's VStitcher as a ground truth and works to create matching garment characteristics which can be utilised in other 3D applications for VR and AR. In our use-cases specifically, these applications are Unity (for VR Designer and MagicMirror) and Lens Studio (for the Snapchat filter). VStitcher is an appropriate ground truth because it is enterprise software designed to be as true to the physical garment as possible. Their technology utilises a sophisticated physics model which utilises scans of fabric samples to understand garment characteristics. It also utilises the garment pattern information to stitch all of the pieces together which makes it as close to the real garment as possible.

Garment simulation software (e.g. in Unity) exists and has many parameters which can be edited. However, it requires manual steps and expert knowledge to make garments look realistic. In most cases, experts will design garments to look good in games, but are not working to match how a specific physical garment would move in real life. Because we are trying to match physical garments in this way, our research is innovative. In this report we will describe in detail how we use a reference garment movement to find and set Unity-Obi particle parameters for simulation in AR. We use optimisation to find the parameters which minimise three defined costs related to the realistic simulation of the garment. Our technique provides a way for more efficient and accurate output than the manual method of particle painting.

MagicMirror is the main showcase of our current cloth simulation optimisation. Using our research we have generated the Unity-Obi parameters for an Odlo t-shirt and have created a Unity application which allows users to try on that garment in real-time. Getting the application fully working required integration of ARKit body tracking which is also described in this document.

As part of the project we also spent some time investigating the quality achievable using existing market leading technology, Lens Studio by Snap Inc. This is important for us to understand the possible exploitation of our technology and also to see the quality achievable by other AR technology in the market. To complete a prototype, we needed to perform various steps to make the VStitcher garments compatible with Lens Studio as well as editing the Lens Studio avatars to improve the quality.

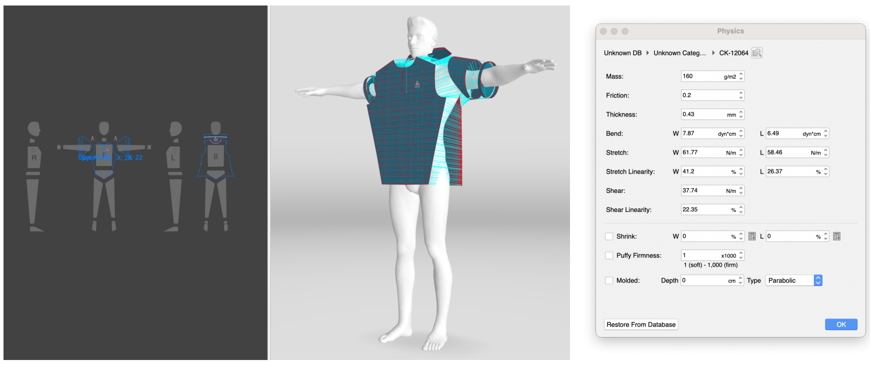
The other use cases are the DressMeUp app and the VR Designer. In both cases so far, the avatar-garment simulation leans more heavily on VStitcher and the work has been focused on the backend and decisions we had to make to reach first versions of working software. For DressMeUp, we will describe the pipeline to go from scanned person to an image with the person wearing the 3D garment. We will also describe a framework we built to assess the quality of the output. For VR Designer, we will describe the requirements of the end-users and describe using animations of avatars and garments directly from VStitcher for this use case.

3. Cloth Simulation

3.1 Physics based garment simulation in Vstitcher

Physics Based Simulation (PBS) is the computer graphic process that synthesizes the shape of a 3D garment model based on predefined physics models. In early research, a spring-mass model was widely used to represent the stretch and compression between garment particles. It has evolved later to include more complex cloth behaviours (e.g. self collision, bending effect, buckling effect) and improve the stability of the system of equations (Terzopoulos 1987, Choi 2002).

Many garment design applications support PBS as a fundamental tool to synthesize how a designed garment will settle on a target body. This process generally involves solving a large system of equations iteratively, so it will require some computation time and the result will be specific to the garment and body. VStitcher also supports garment simulation from 2D cutting patterns with a range of fabric physics parameters to deliver realistic PBS results. The simulation process is similar to many other designer applications, which includes a) initial garment arrangement around body parts, b) stitching patterns in 3D and c) running the simulation iteratively (see figure 1). It is also worth mentioning that the body model should be free from the body self-intersections to avoid long simulation or any unwanted simulation artefacts such as minor body clippings. Initial garment position can play an important role to produce a stable outcome as well.



Mass:		160 g/m2		
Friction:		0.2		
Thickness:		0.43 mm		
Bend:	W	7.87 dyn*cm	L	6.49 dyn*cm
Stretch:	W	61.77 N/m	L	58.46 N/m
Stretch Linearity:	W	41.2 %	L	26.37 %
Shear:		37.74 N/m		
Shear Linearity:		22.35 %		
☐ Shrink:	W	0 %	L	0 %
☐ Puffy Firmness:		1 x1000 (1 (soft) - 1,000 (firm))		
☐ Molded:	Depth	0 cm	Type	Parabolic

Figure 1. Garment PBS in VStitcher: 2D cutting patterns are arranged around the associated body parts (left) before running PBS, which will find the new settling position of a garment using many textile physics parameters (right)

The static PBS can be easily extended to create an animated garment, e.g. we can cache the PBS result at each keyframe and play them continuously later. One main difference might be that an animation can reuse the result from the previous frame as an initial garment position and we can also feed the dynamic information (e.g. inertia) if required. In this project, we generate animated garments and body models as the main assets for the UC1 designer app (Sect. 7), and we also use them as a reference point in the UC3 when we optimise the garment particle values (Sect. 5.2).

Creating a reference garment animation starts from preparing an animated body model. To facilitate this process, we develop a new animation baking pipeline. Since body animation is a smooth change of 3D pose parameters (i.e. a list of 3 relative rotation values at joints), we can theoretically produce any animation by defining pose parameters at keyframes and interpolating the values over time. However, producing a credible body animation is another problem, which is tricky to do manually as the result should account for the appropriate kinematic body constraints from different body shapes. Instead, we extract the motion data from the open-source body animations in Adobe's Mixamo.com. The proposed pose baking pipeline transfers a list of Mixamo body animations to our body model, inserts a canonical pose (e.g. A-pose or T-pose) at the beginning of the animation, and then interpolates between keyframes. (see figure 2).

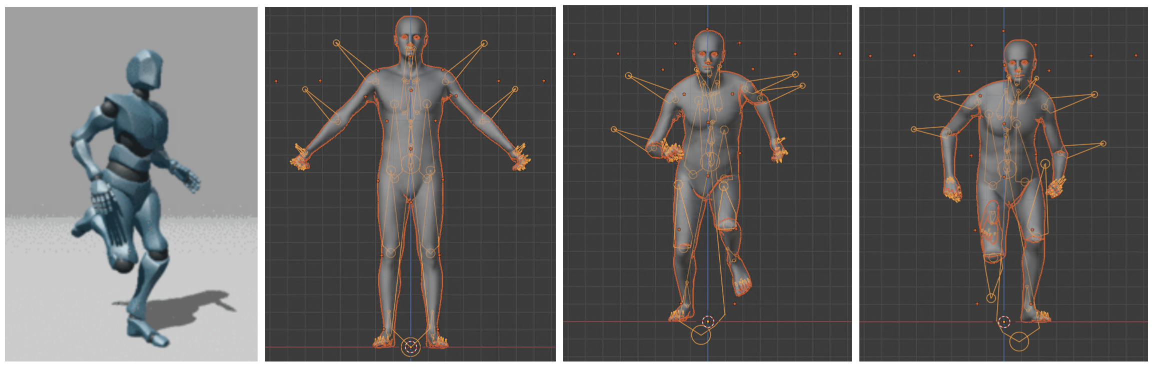


Figure 2. Baking body animation: creating animated garments starts from preparing valid body animation. The proposed pipeline transfers the animation from an open-source library (first image) to a target body model with a canonical initial keyframe pose (second image). Transferred running animation at keyframe 10 and 20 are also shown in third and fourth image, respectively.

There is no single, standardised data format for animated 3D models that supports all the 3D information we need to support. The challenging point is how to efficiently store a large amount of data from a moving 3D object in a complex scene graph. The most popular format is FBX with Maya cache files. This is the same as FBX format for static 3D models but with additional cache files storing the vertex changes in each frame. Recently, alembic file format (ABC) is also widely used in many visual effects, 3D animations, and games as an alternative to the proprietary FBX data format. We also use ABC file format to store the reference animation efficiently as well as static FBX file format to store the UV mapping and skinning information.

3.2 Real time garment simulation in game engines

PBS-based garment simulations normally produce the most realistic garment draping results. However, the computational load involved in the PBS makes it almost impossible to use it within any real-time 3D applications. This will be a critical limitation for VR/AR apps, which normally need to deliver more interactive simulation results rather than realistic simulations, such as deforming the garment according to the tracked body pose, or estimating collision with a moving body underneath.

This section will explain two fundamental rendering tools (such as skeletal animation in Sect. 3.2.1 and particle based animation in Sect. 3.2.2) that we used for developing responsive garment rendering pipeline for better AR experience.

3.2.1 Skeleton based simulation

As a workaround for the timetaking PBS, many 3D applications adopt skeleton-based animation for garment rendering. Skeleton-based animations (also known as “skinning animations”) basically embed a skeleton in a target 3D model and estimate new vertex position from the underlying skeleton movement. For example, “linear blend skinning” assumes that the new position of the surface vertex can be estimated by a linear combination (i.e. a weighted sum) of bone transforms. This means we can compute new garment positions from a few matrix multiplications (e.g. Unity normally uses only 4 associated bones in their humanoid object animation), which is a significant reduction of computational load compared to what is done with PBS in an iterative way.

However, the skeletal animation has the following practical issues:

- 3D designers need to define **correct skinning weights** for each surface vertex (which will be internally stored in a mxn matrix, where m is the number of vertices and n is the number of bones). This process is normally done by vertex painting which is not trivial for novice users.
- In addition, each bone of the skeleton should be correctly placed within the body. Incorrect bone positions could result in body interpenetration even with the valid body pose parameters. This will be noticeable especially with 3D body models with high BMI or with extreme motions. This makes 3D content creators spend a long time to **adjust bone positions** and the associated skinning weights.

The process of defining skinning weights and placing bone positions are often collectively called “rigging” or simply “skinning” (so it should not be confused with “skinning” in the skinning animation).

In our AR application for real time garment simulation (e.g. MagicMirror app), we also adopt skeleton based animation as a baseline garment rendering tool. To avoid the manual skinning process, we assume that a garment model shares the same bone structure with the underlying body model. This seems to be reasonable as the garment is normally designed to wrap around different body parts. To simplify the process further, we also assume that the garment shares the same skinning weights from the nearby body vertex (see figure 3).

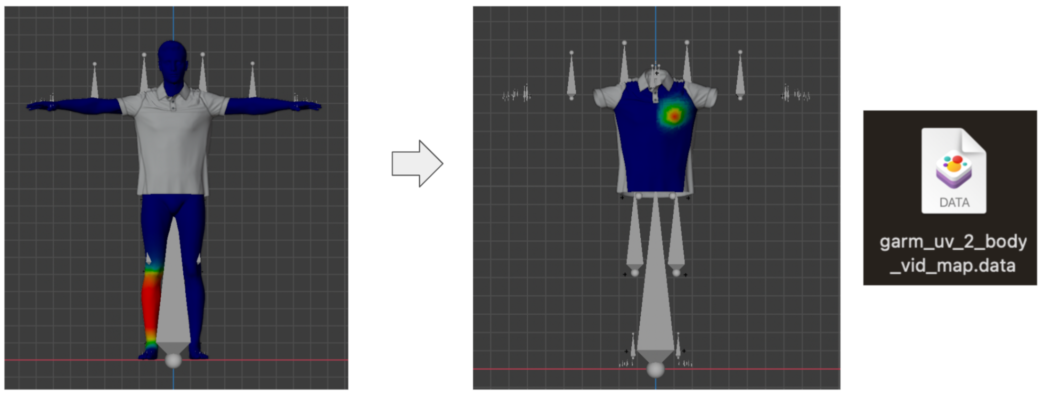


Figure 3. Garment skinning at T-pose; this is similar to the previous garment skinning process we used for our Snapchat filter. One difference this time is it will store the body-to-garment vertex mapping table to skin the garment at an arbitrary future pose.

The skinning process developed in this period is similar to the previous garment skinning we have developed for a Snapchat filter. But the idea was slightly extended this time to store additional vertex mapping information for skinning a garment at an arbitrary position (see fig. 3.4). This will be especially useful when we need to compare the rendering result from a reference animation in ABC format with the animation result from a rigged FBX in Unity.

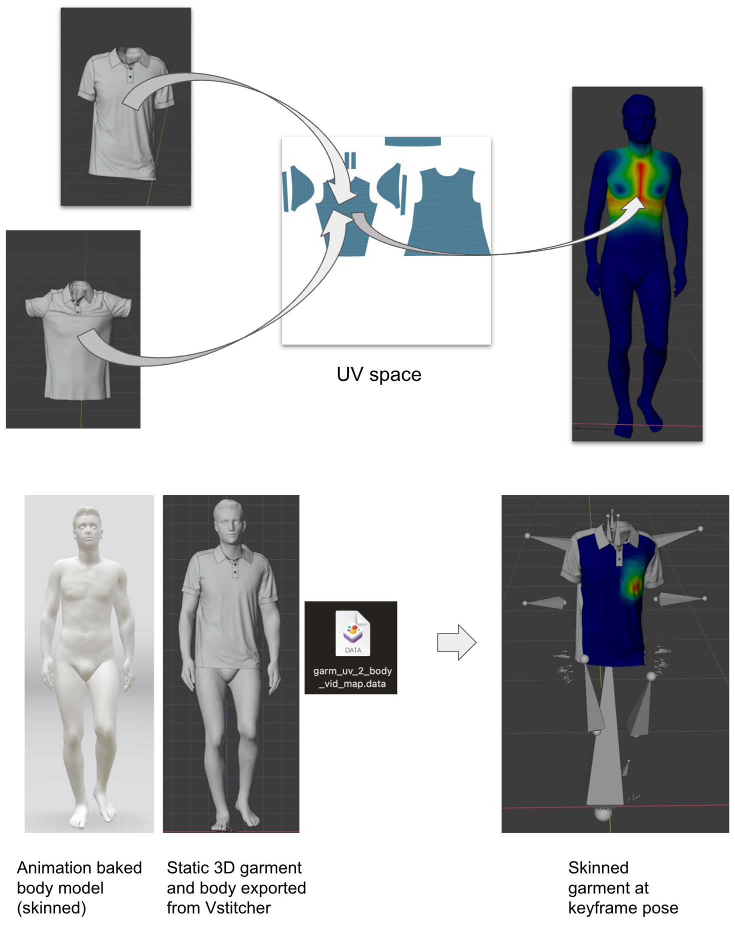

UV space
DATA
garm_uv_2_body
_vid_map.data
Animation baked body model (skinned)
Static 3D garment and body exported from Vstitcher
Skinned garment at keyframe pose

Figure 4. Example of garment skinning at an arbitrary body pose; transferring the skinning weights from the nearest body vertex to a garment vertex is not ideal with some body pose (e.g. the weights for the long sleeve can be transferred incorrectly from the torso rather than the arm). To avoid this confusion, we define a body-to-garment vertex mapping at T-pose, store the table in a UV map (top) and then reuse it for skinning transfer at an arbitrary body pose (bottom)

3.2.2 Particle based simulation

The skeletal animation can deliver fast garment animation. However, the result is generally not realistic enough. This is mainly because it cannot depict local shape deformations over time. Since this nonlinear behaviour is hard to capture properly with a weighted sum of linear transforms, there is a certain limitation that we can achieve from the skeleton based animation; e.g. the resulting garment looks glued on a body with fixed creases.

As a solution for this, we designed a rendering pipeline that applies particle based simulation on top of the skeletal animation for our MagicMirror app (see figure 5). Due to the efficient physics computation with sparse particles, the proposed pipeline realises fast local deformation without incurring too much computation.

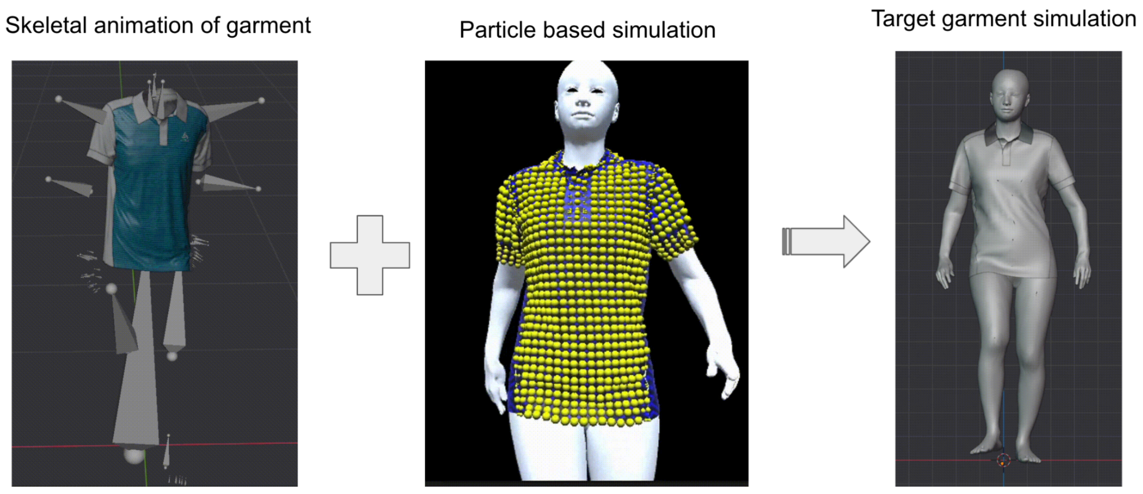

Skeletal animation of garment
Particle based simulation
Target garment simulation

Figure 5. Our garment animation uses particle animation on top of the skeleton animation of a garment to capture the dynamic local movement of a garment

The particle based simulation minimises the heavy physics computations by approximating a target garment with a sparse set of particles distributed evenly across surface meshes (see figure 5. middle). An idea of using particles and the constraints between them to simulate physics effects is quite similar to the classic spring-mass model. However, the latest particle models not only generalise it further for any soft body objects (e.g. gasses, fluids, ropes, and garments) without requiring additional complex nonlinear models, but it also implement novel particle physics solvers which can parallelise the process (Macklin 2014).

Nvidia Flex is one of the good candidates for the particle garment simulation. However, we found that is not the best option for us as it cannot handle the multiple garment submeshes from VStitcher without additional preprocessing steps. Although it is available from Unity, our initial tests also found that setting up the particle parameter values is not trivial and it is difficult to make the result match to our reference animation. As an alternative, we use Obi-cloth which can provide similar functionalities with more flexible interfaces to particle properties and has the ability to stitch multiple submeshes.

At the time of writing, most particle simulation libraries do not support mesh colliders if particle physics is running with skinning animation. This seems to be a reasonable design decision to avoid long computational time for real-time collision detection. However, this means the users need to set particle parameters correctly to avoid body clipping. One hybrid solution could be configured to address this, e.g. approximating the body with a set of primitive colliders (e.g. cylinder colliders). This idea was explored early in this period but in the end we decided that the automatic creation of a body collider was unnecessary. Body collision is no longer a significant problem in AR because the user's body is used and the image compositing happens separatately to the physics simulation.

3.3. Data driven garment simulation

Another approach we can try to improve the quality of garment simulation is data-driven garment rendering. This approach is getting more attention recently due to the significant performance improvement via various new deep learning (DL) techniques. Many researchers are now even challenging classic forward rendering pipelines with different DL models trained from large data samples of partial information; e.g. neural radiance field can render an image at arbitrary viewing position after training only with multiple 2D images (Mildenhall 2020, Barron 2021).

In this project, we are also interested in exploring the potential of deep learning solutions to overcome the limitations from the classic garment pipeline explained in Sect. 3.2, and produce results that can be comparable with the result from the classic PBS explained in Sect. 3.1. This machine learning approach will be different to the models for "Neural Rendering", in a way that it will focus more on predicting better ***garment geometry*** from dynamic body motion rather than delivering ***photorealistic images***. We call this model Neural Dynamics (ND) similar to Neural Rendering. The following subsections will explain more details about our plans and summarizes the current status of this stream of the work in the project.

3.3.1 Neural dynamics

Data-driven garment simulation is a well-known problem in the computer graphics community. One of the early works closely related would be DRAPE (Guan 2012), which extends the idea of reconstructing nonlinear body shape using principal component analysis (PCA) to predict nonlinear garment deformation from body shape and pose parameters. Since DRAPE solves the problem by optimizing the cost to find the best PCA coefficients for the deformation transforms, it is not the same as the recent DL solutions. However, the ideas behind DRAPE are still valid in many latest garment deformation models (e.g. interpenetration loss, garment damping terms).

Similar to DRAPE, our ND should be designed to predict a nonlinear garment shape deformed by body shape and pose parameters. More specifically, we would like to predict the vertex offset between the actual deformed garment and the one from the skeletal animation. This incremental rendering approach can help reduce the range of predicted values and will make ND more compatible with the conventional rendering pipeline.

We are expecting the proposed ND will perform better in terms of the processing time and the fidelity of the deformation. However, we also foresee the following technical challenges;

- 3D data representation
  Unlike 2D images, ND should take as input 3D models. It means the ND architecture would change significantly depending on the data representations (e.g. 3D point cloud, volumetric data, implicit surface function, and UV map). One noticeable difference from 2D images is that the number of the local neighbours can differ at each vertex for 3D data. This makes it difficult to reuse the conventional CNN. As an alternative, we can adopt Graph CNN (GCN). However, the complex operation of GCN could be a potential issue for developing an ND model for mobile environments (e.g. Lens studio or Spark AR studio). The feature pooling operation will be more elaborated as well. Although there are various GCN/pooling ideas, it is hard to tell which would work best for different applications.
- Model generalisation
  At the time of writing, most garment draping models are garment specific (Gundogdu 2019, Patel 2020, Saito 2021). It means that a trained model can predict the deformation from different body poses and shapes, but we need to train it for each garment. It is probably natural, considering the large shape variations between different garment types. However, we hope we can generalise it at least for different garment sizes within the same garment. It should be also noted that GCN requires a template garment to ensure all training samples have the same topology. Thus, it makes it more difficult to develop a generalised ND model. There are some latest models (Ma 2021) that can generalise the different garment types using the latent vector from auto-decoder, but it requires a different 3D data representation using a UV map.
- Deterministic behaviours of supervised model
  Local creases found on garments are not deterministic with respect to body parameters, but vary with how the garment is dressed on the body and previous movement of the body and garment. If we train a model in a supervised manner, the deformation results will be fixed for specific input body parameters. In order to synthesize the randomness of the creases, some researchers have tested

various generative models (Lahner 2018, Ma 2020). The downside of using a generative model is that training will be more difficult and unstable.

- DB and data preprocessing
  To train a DL model, we need a large number of data samples. The number of required data samples can be different depending on the unknown parameters in your deep learning model. But a general rule of thumb is a few hundred thousand samples are normally required to train/test a model. Collecting this amount of 4D data from PBS will be labour intensive and time taking. It also requires many 3D data preprocessing tools (e.g. rigging, animation, skinning, and reposing), which could be another overhead for developing ND models. We will explain more details about DB in the following sections (Sect 3.3.2).

Amongst recent DL models, we found GarNet (Gundogdu 2019) will be a good starting point to build our ND model. Since GarNet adopts PointNet (Qi 2016), it can take an unordered point cloud as input and extract features in a transform invariant way. We are particularly interested in the PointNet architecture for segmentation, which was designed to combine local features with global features to create a per-vertex segmentation score. We think this feature fusion will be recycled in our per-vertex deformation estimation. Another important feature of PointNet is it does not use topology, so we do not need to bind with a specific template garment topology. Fig. 3.6 shows an example pipeline for a GarNet based ND model. We could improve the model in future to fuse the local features from body and garment more explicitly.

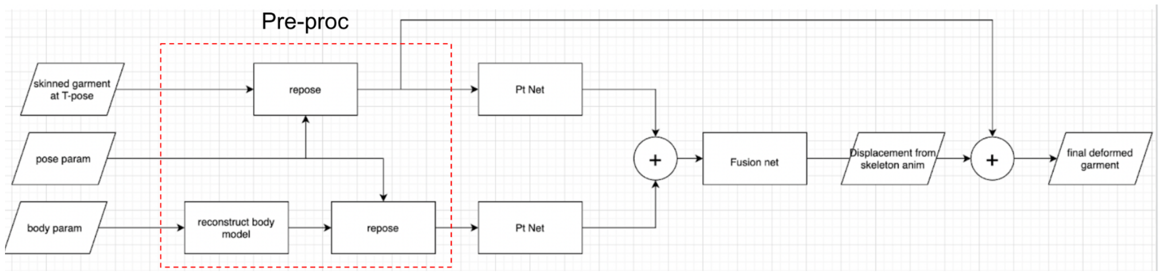


Figure 6. example of GarNet based garment deformation model

In addition to the general risk for ND we assessed earlier, we envisage that a GarNet based ND model will suffer the following issues:

- Body-garment interpenetration
  Although there are specific loss terms for improving body clipping in GarNet, we think the missing pose normalisation step in GarNet would give an incorrect estimation resulting in body clipping for some body shapes. This problem is known from early research and many suggest using additional post-optimisation to correct the inferred deformation offsets (Santesteban 2019, Guan 2012). There is some new research that tackles this clipping problem with a novel DL architecture (Santesteban 2021). Alternatively, we can simply measure the deformation from the corresponding body vertex to address the issue (Ma 2021, Ma 2021b)
- Local details

Many deep learning models try to be general so that it can give a reliable result from diverse inputs. A problem with this is that the final model is not able to capture the fine details at the cost of the generality of the model. To address this, many garment draping models have a separate subnetwork to learn this discrepancy (Patel 2020, Santesteban 2019). At this stage, we plan to develop a simpler model that works for mobile environments, so we will try other network techniques to improve base GarNet.

- Deterministic details
  As mentioned earlier, this will be inevitable with supervised learning. This is not the higher priority at the moment and we rather focus more on the general deformation of a garment to deliver the fit related information for MagicMirror.
- No dynamic features
  The draft model shown in fig. 3.6 does not use any dynamic information from motion. As other research suggested, it would be useful to exploit the dynamic features from the previous frames. We think LSTM or BERT style networks (Devlin 2019) would be useful to learn the dynamic features. However, this will not be a high priority at the moment because it is not as important for fit prediction.

3.3.2 Garment DB for Neural Dynamics

In order to learn garment deformation from different body motions, we require a variety of 3D data, including rigged body models, compatible motion data (i.e. pose parameters), and garments. They should be appropriately augmented to span our target application space.

There are some existing DBs useful for learning garment deformation (see Table 3.1), but most of them are not open to the public or only available for research purposes. Most importantly, since each DB is created for specific applications they are not the best DB for training our ND model. To address this, we have developed a new animated garment DB using PBS.

Table 3.1. Existing garment DBs developed for garment animations

	Data source	Body model	Motion	Garment
DRAPE	PBS (by Optitex)	CAESAR (avg M/F), SCAPE (60M+60F)	~20 motion seq.	5 graded garments from 2D patterns using optitex
Deep wrinkle	4D scan	Single body (SCAPE)	Unknown (from 4D scan)	Single T-shirt (~5K vert)
GarNet	PBS (by NvCloth)	600 SMPL body models	60 motions from CMU Mocap DB	3 garments (~10K vert for each); T shirt, sweater, jeans
CAPE	4D scan	11 ppl w/o and w clothes, registered w SMPL	CAPE motions for each scan	8 garments

TailorNet	PBS (by Marvelous designers)	9 SMPL bodies	1982 static SMPL poses	Synthetic garment samples generated from garment PCA, followed by PBS
Learning based animation	PBS (by ARCsim)	17 SMPL bodies	56 CMU motion seq. (~7K frames)	Single T-shirt (~9K vert) and use retargeting technique for diff size
SCANimate	4D scans (CAPE)	CAPE bodies (SMPL)	CAPE motions, ASIT++, AMSS	CAPE (real 4d scan of clothed subject)
Self-supervised collision handling	PBS	17 SMPL bodies	AMASS	Dress, T-shirt
Cloth3D	PBS (by Blender)	100 SMPL	23 actions from CMU motions	4 garments + synthetic variations (shape, material, tightness)

Our new DB has been built in a similar way to Cloth3D (Bertiche 2021). For example, the reference garment drapes are generated using VStitcher from 50 animated body models (i.e. 32 males and 18 female models from actual scans) which were animated by 25 motions from Mixamo. Our garment models are created from 2D cutting patterns and we collected them from open source libraries and internal garment DBs (see figure 7, top). Each garment has multiple sizes which will be used to populate different garment samples within the same type for future research. More information about our DB is summarised in Table 3.2.

Table 3.2. Summary of Metail garment DB for ND

	Data source	Body model	Motion	Garment	Tot no. anim.
Metail DB (set1)	PBS (by VStitcher)	2 body models (M1 and F1) from actual scans	Each body embeds 25 Mixamo animations	8 Garments from 2D cutting patterns	400
Metail DB (set2)	PBS (by VStitcher)	48 body models from actual scans (M31/F17)	Each body embeds 25 Mixamo animations (this can be different motion from set1)	8 Garments from 2D cutting patterns	84,000

Some of the sample animations we have created (in ABC format) are found in figure 7, bottom. The generated animations often suffer undesired artefacts. For example, the garment would be falling on a wrong settling position because some parts of the garment were caught by hands or head. It is also possible that the simulated animation could create significant body intersections even though pose parameters are valid. These samples are filtered out manually using separated QC tools. Some sample gif files generated for our manual QC are also shown in figure 7, bottom right.

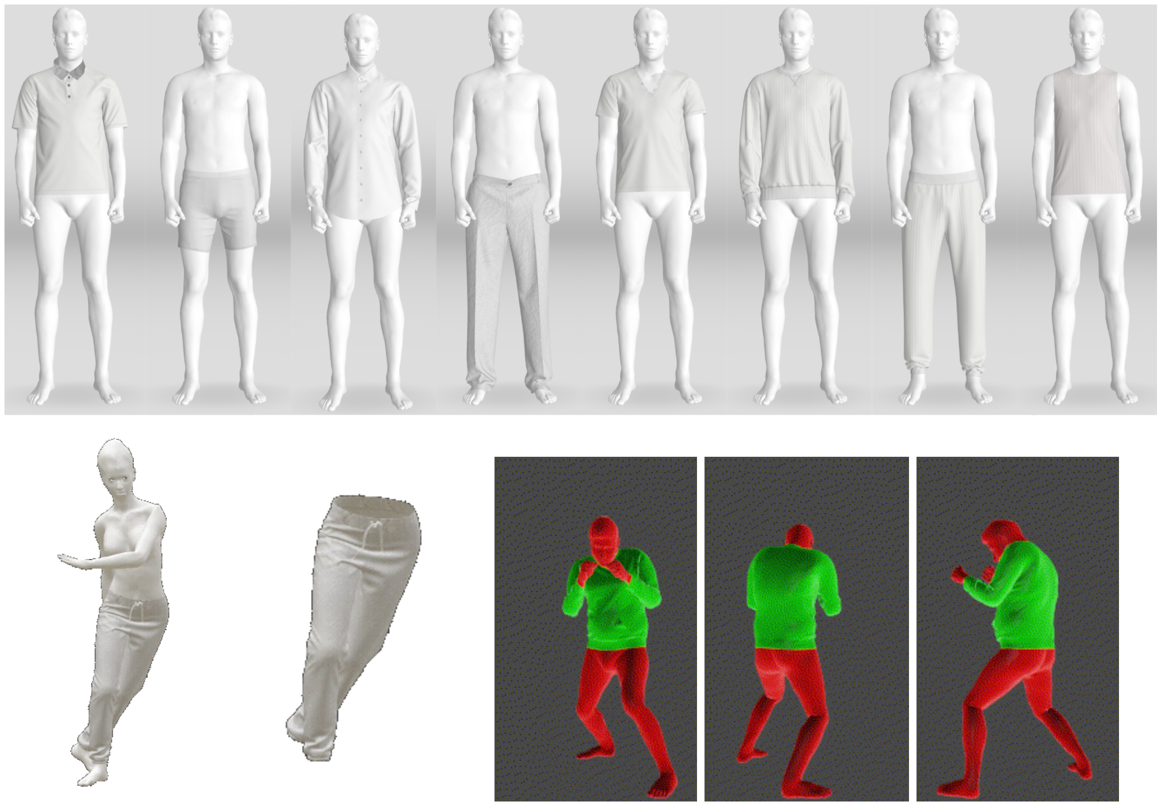


Figure 7. Example of garments used in our DB (top), sample animation (bottom, left), GIF images from 3 different viewing angles automatically generated for manual QC process (bottom, right)

4. Snapchat Filter

Lens Studio by Snap Inc. is a framework designed to build augmented reality experiences for Snapchat. Because it has many built-in features, including custom shaders and body tracking, and can be used to create AR experiences for an existing user audience of over 300 million, we have used it to demonstrate the goals of the Magic Mirror app (use case 3).

So far, we built our own avatar, based on average body shapes, and created tools to create a skeleton compatible with Lens Studio. However, by dressing this avatar we produced results poorly aligned with the user's body. See some of the examples below.

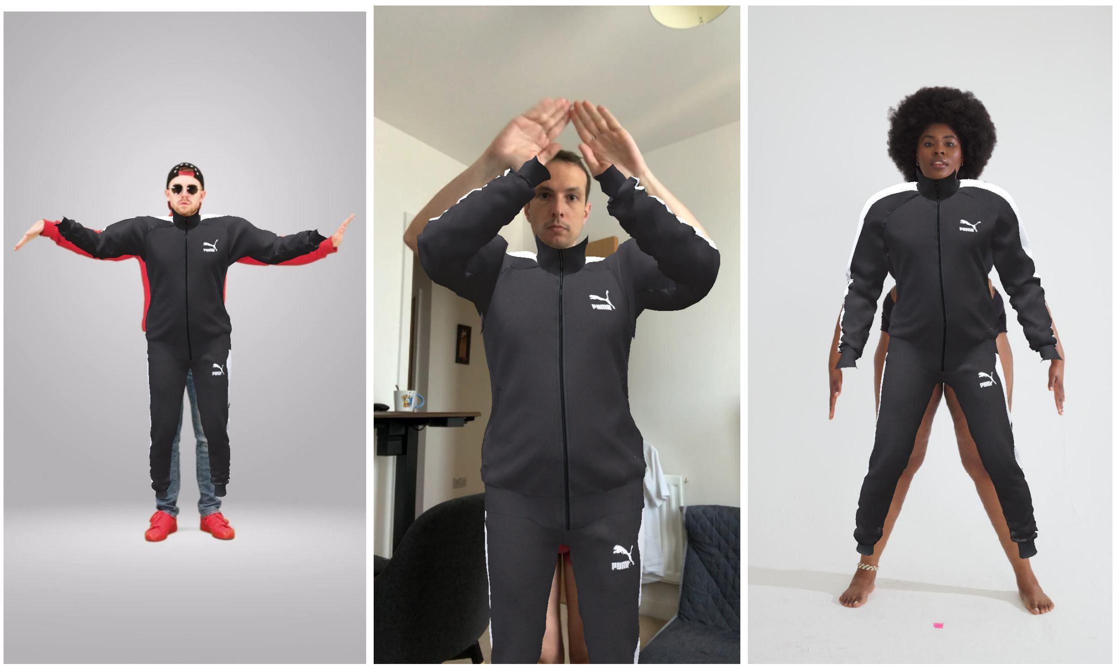


Figure 8. Some examples of a tracksuit Snapchat filter, where the garment was simulated on an average male body shape.

4.1. New avatars for Lens Studio (Rens)

Snap provides a neutral avatar in a T-pose as reference. We decided to use that avatar in our pipeline to try to better the alignment with the user's body in AR. The file provided by Snap is an FBX file, with the correct skeleton for Lens Studio, but we needed to manually add the anchor points for dressing in VStitcher and an A-pose so we can dress the avatar in a more natural way. We called this avatar "Rens" version 0. The figure below shows Rens next to the average male body shape we had used earlier.

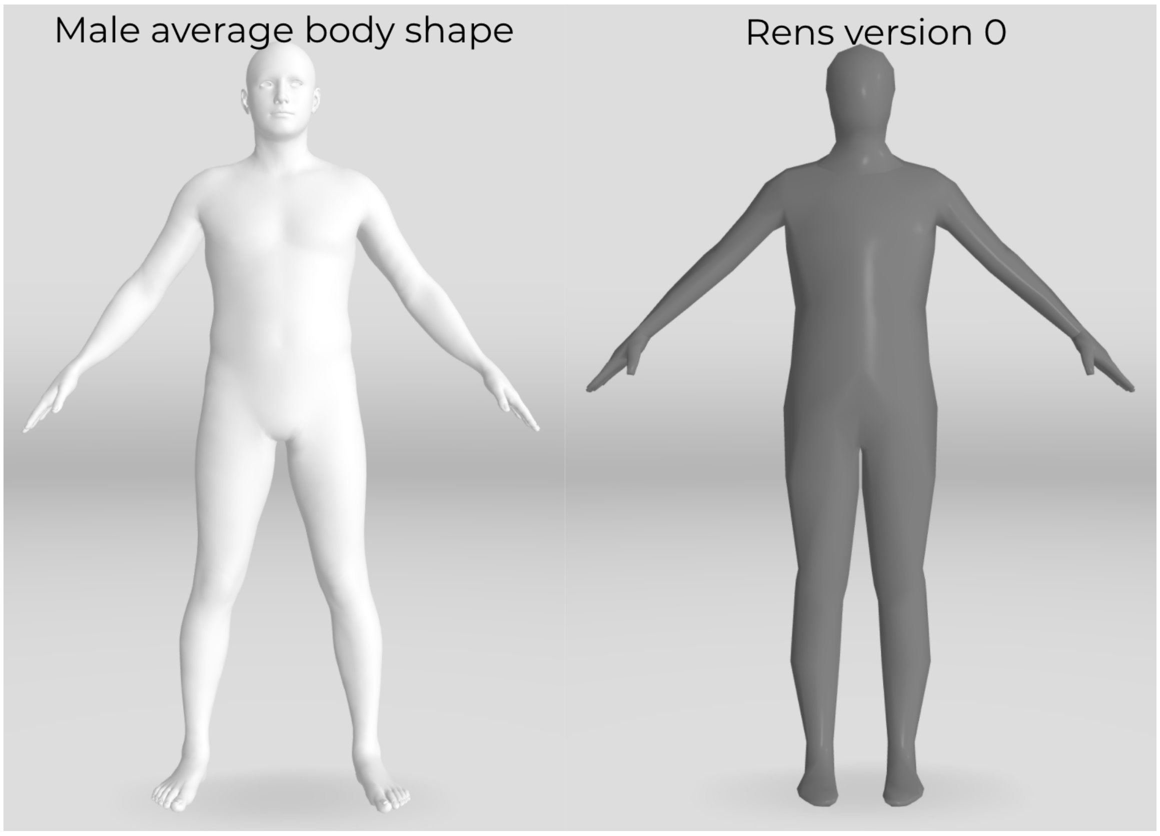


Figure 9. Average male body shape (left) vs Snap body mesh (right).

We have also developed a plugin for VStitcher to easily export the garments and make them ready to use in Lens Studio. So the steps to create a Snapchat filter are:

1. Dress Rens & reduce the grid size of the simulation (Snap recommends a maximum of 60K triangles per scene).
2. In the plugin, select “Export for Lens Studio”. This automatically exports the FBX file, removes trims and buttons with a high number of polygons, generates all the necessary textures, and skins the garment based on Rens avatar. The skeleton is simply copied over from Rens.
3. Load the skinned garment FBX in Blender and save it again (this is a workaround because Lens Studio can’t read our files directly for some reason).
4. Drag & drop the skinned garment FBX in Lens Studio and import all the textures and materials.

As you can see in the figure below, this avatar already produced better alignment with the user’s body.

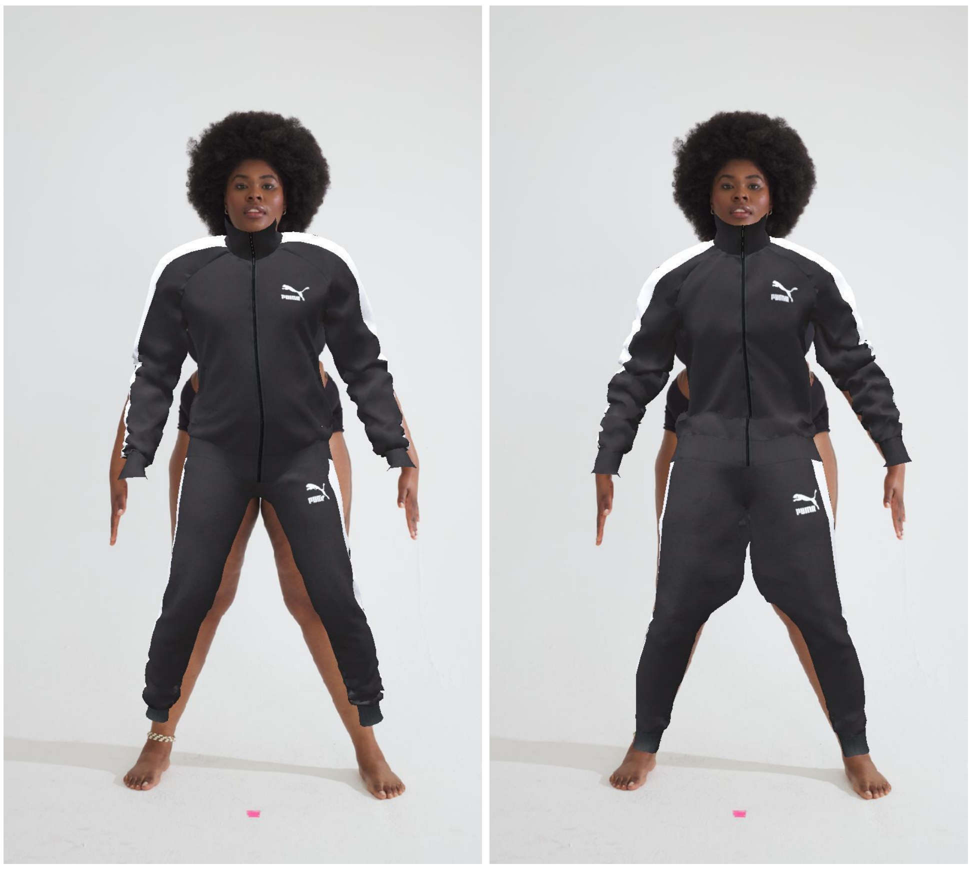


Figure 10. Comparison of the tracksuit Snapchat filter using the average body shape (left) vs using Snap body mesh (right)

However, we have introduced another problem around the legs. As you can see, the trousers get pulled towards the opposite leg as you get closer to the crotch. This artefact happens for 2 reasons:

1. When we assign a skinning weight to a vertex in the garment, we search for the nearest vertex in the avatar. If the garment is a bit loose, in places near the crotch the nearest vertex may end up being on the other leg. A similar problem happens under the armpit, where the closest vertex of the sleeve could end up on the chest instead of the arm.
2. To smooth out the skinning, we take the K-nearest vertices and average their skinning weights. But the tessellation of the avatar provided by Snap is very low, so the second nearest vertex for a point on the thigh could end up being at the knee, since there aren’t any more vertices in between.

To resolve those problems we created a new version of Rens with increased tessellation, and with an A-pose with the legs further apart. The tessellation was increased manually by editing the mesh in Maya. The tessellation of version 0 is compared with the latest version below.

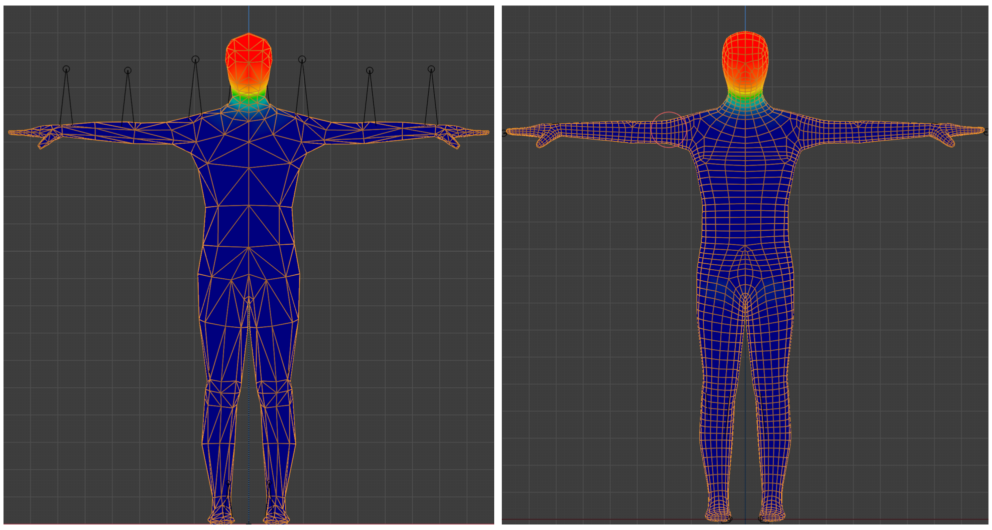


Figure 11. Original tessellation in Snap body mesh (left) vs our retopologized mesh (right)

The pose selection at the moment is a manual process that depends on the type of garment. We found out that for skirts, the legs closed work better, because the pose looks more natural. But for trousers, we better use this new pose with the legs further apart to avoid capturing skinning weights from the opposite leg, as described above. The figure below shows the latest version of Rens, and how the garment looks once dressed in VStitcher.

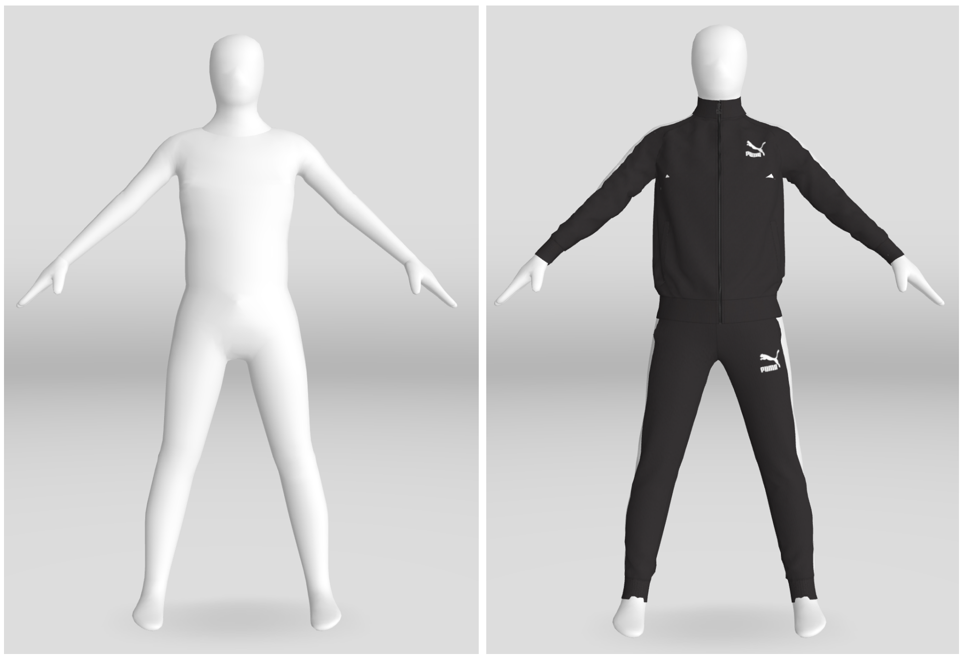


Figure 12. Our Rens avatar with increased tessellation in VStitcher.

The figure below shows some results with the new version of Rens.

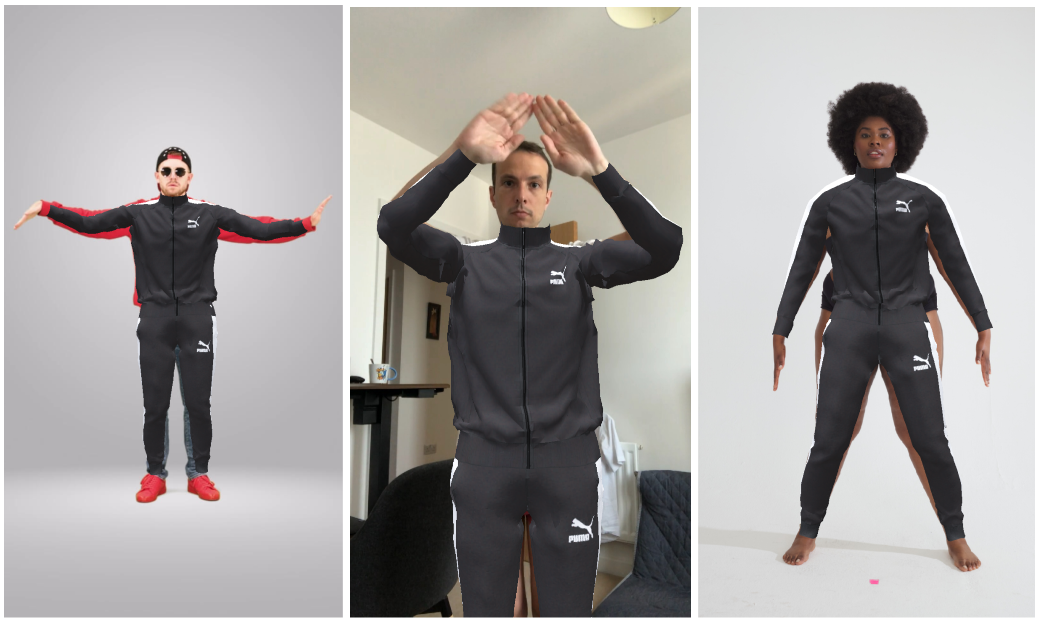


Figure 13. Some examples of a tracksuit Snapchat filter, where the garment was simulated on the latest Rens avatar.

4.2 Avoiding Z-fighting

When the garment is tight, the position of the triangles of the garment is very close to the position of the triangles of the avatar. When rendered from a distance, the depth values of the individual pixels inside each triangle may randomly appear in front or behind the avatar, depending on the precision of the depth buffer. This phenomenon is called Z-fighting.

In AR we may not need to worry about Z-fighting if we only render the garment. However, the avatar is usually rendered as well with a special occluder material, so the parts of the body can occlude the garment. The figure below shows a bad case of Z-fighting with a full-body occluder. Note that most of the top disappears.

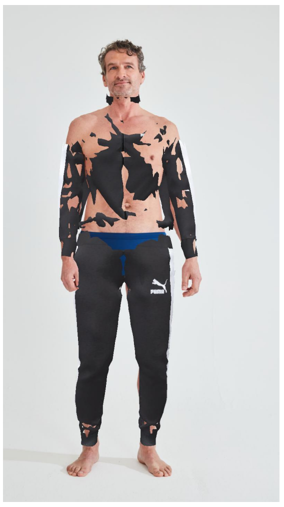


Figure 14. Bad case of Z-fighting, caused by the triangles of the occluder mesh sharing positions with the garment mesh.

In reality, there are two problems that we conflate here under the label of “Z-fighting”, genuine Z-fighting because of very similar depth values, and misalignment problems that cause the garment to end up inside the body.

There are 2 main causes for misalignment. The first one is due to a global offset that VStitcher applies to the models when imported, based on the crotch and feet positions. We can compute this offset and undo it during export. Our VStitcher plugin takes care of that.

The second cause of misalignment doesn’t have a simple fix, though. It is caused by skinning itself. The garment is only perfectly aligned to the avatar in the pose that we dress the avatar in, which in practice, we will never see in the real application. The user will always move and appear in a different pose. And the body shape is also different. Because the skinning is applied to the garment separately, and the topology is also different from that of the avatar, triangles can easily cross.

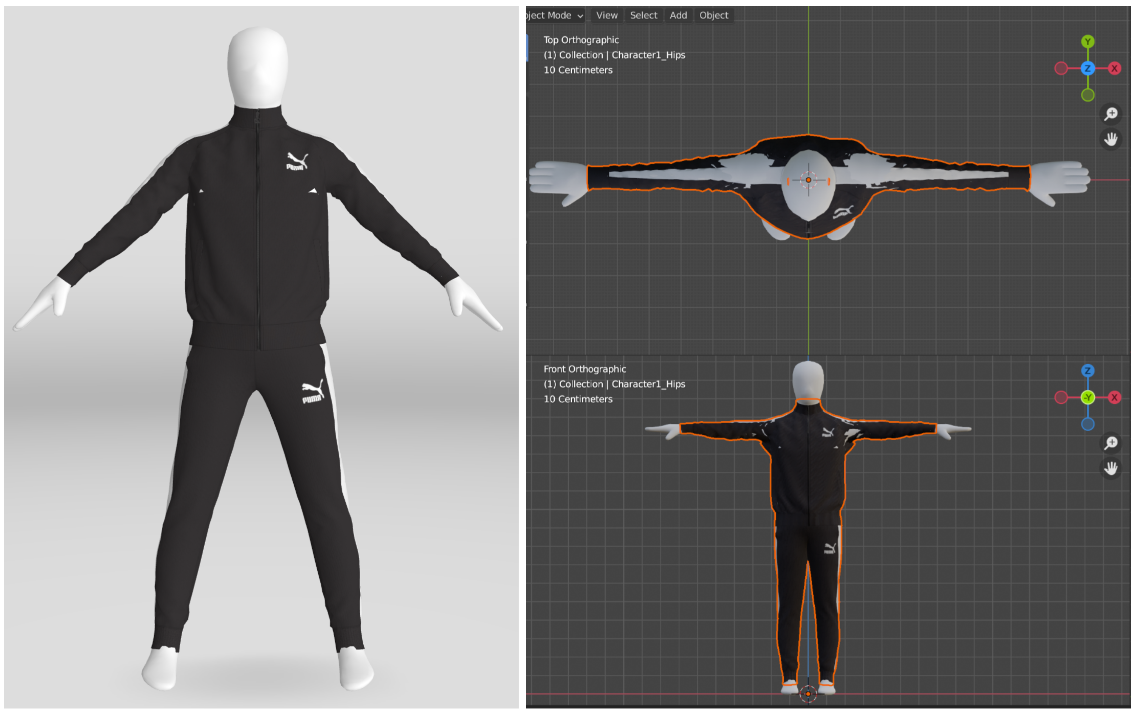


Figure 15. Tracksuit dressed in A-pose (left) and then reposed after skinning to a T-pose (right). Note that some triangles intersect the arms.

To try to alleviate this problem we could create a version of Rens avatar slightly wider, and create a bit of gap with respect to the avatar in Lens Studio. But that doesn’t resolve all cases, unless we exaggerate the difference a lot. But then a tight garment wouldn’t look tight anymore.

We opted for creating our own occluder body for Lens Studio which is split into different body parts. For instance, for a full-body tracksuit, we enable the head, neck, and hands occluder. Since for arms, torso, and legs there is no occluder mesh, Z-fighting won’t occur. The inconvenience of this approach is that the occluder needs to be set up independently depending on the type of garment. For some garments it may be ill-posed. For example, if we have some trousers full of holes, ideally we would like an occluder that only covers the parts where there are holes in the garment, but that’s not possible to generate in the general case.

4.3. Demo - Lens using new Odlo garments (Scott MTB)

You can find in Instagram a promotion video of the released Snapchat filter, with Odlo's Scott MTB T-shirt.

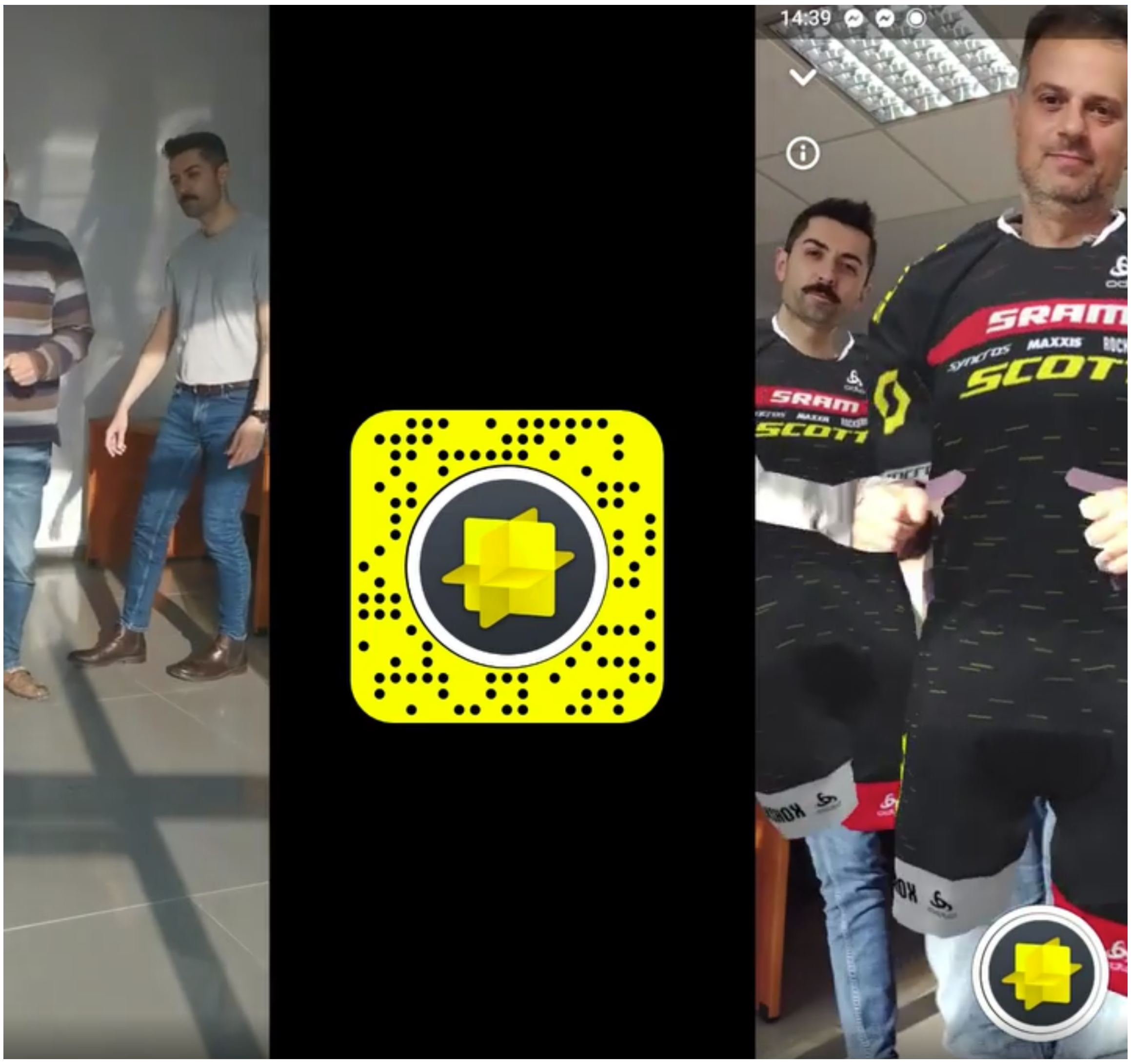


Figure 16. Video of Odlo garment virtual try on Snapchat filter

https://www.instagram.com/p/CWnxzlqrfkP/

5. Magic Mirror

5.1 Overview of the application and required backends

Magic Mirror app is targeting online shoppers who want to see the 3D garment they are thinking to purchase on their body. Thus, it needs to provide the required information by simulating the physical experience of try-ons in the digital space. For this, we rely on game engines, i.e. Unity and the following backend technologies:

- Fast yet realistic garment rendering
- Real time body tracking
- Garment image compositing

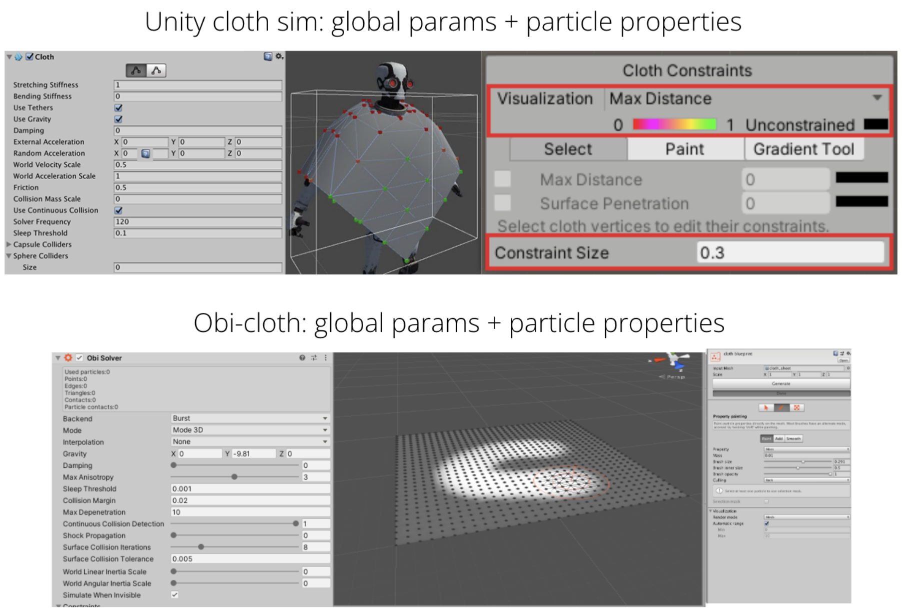
Since our image compositing ideas developed in the Snapchat filter (Sect. 4.2) can be reused to tackle the garment compositing in Magic Mirror AR, the following sections will focus on explaining the technical challenges for garment rendering (Sect. 5.2) and real-time body tracking (Sect. 5.3).

5.2 Cloth sim optimisation

As mentioned in Sect. 3.2.2, our approach for the real-time rendering of a garment in an AR app is developed based on particle simulation with garment skinning animation. The particle-based physics simulation has proven its real-time performance and the fidelity of the rendering soft bodies (Macklin 2014). However, it is not intuitive for novice content creators to configure the physics parameters.

In general, most physics parameters can be played as global variables for garment simulation (e.g. friction and bending/stretching stiffness for a garment), so that a single constant value would suffice to simulate some garment effects. However, manual adjustment of multiple values will be daunting at times, and the simulation results are often too sensitive for small changes.

For expert users, it is also possible for more granular adjustment. For example, many cloth simulation plugins provide vertex painting UI to restrict the local movement of a garment (see figure 17). It allows users to control the detailed movement in addition to the general shape deformations from the global physics parameters.


Unity cloth sim: global params + particle properties
Obi-cloth: global params + particle properties
Cloth Constraints
Select
Paint
Gradient Tool
Select cloth vertices to edit their constraints.

```
Cloth
Stretching Stiffness          1
Bending Stiffness             0
Use Tethers                   ✓
Use Gravity                   ✓
Damping                       0
External Acceleration         X 0    Y 0    Z 0
Random Acceleration           X 0    Y 0    Z 0
World Velocity Scale          0.5
World Acceleration Scale      1
Friction                      0.5
Collision Mass Scale          0
Use Continuous Collision      ✓
Solver Frequency              120
Sleep Threshold               0.1
▶ Capsule Colliders
▼ Sphere Colliders
    Size                      0
```

```
Visualization   Max Distance
0   1   Unconstrained
Max Distance          0
Surface Penetration   0
Constraint Size       0.3
```

```
Obi Solver
Used particles:0
Points:0
Edges:0
Triangles:0
Contacts:0
Particle contacts:0
Backend                          Burst
Mode                             Mode 3D
Interpolation                    None
Gravity                          X 0    Y -9.81    Z 0
Damping                          0
Max Anisotropy                   3
Sleep Threshold                  0.001
Collision Margin                 0.02
Max Depenetration                10
Continuous Collision Detection   1
Shock Propagation                0
Surface Collision Iterations     8
Surface Collision Tolerance      0.005
World Linear Inertia Scale       0
World Angular Inertia Scale      0
Simulate When Invisible          ✓
```

Figure 17. Example user interfaces for particle-based garment simulations; Unity cloth simulation (top) and Obi-cloth (bottom)

In Obi Cloth, we identify three per-particle parameters (i.e. skin radius, backstop, and backstop radius) useful to control the local deformations (see figure 18). These parameter values can be stored in a UV map in an Obi-blueprint with additional metadata. However, for some novice users (even for some mature 3D artists), preparing these per-particle values would be challenging. We can easily see that the process is overly simplified with a single fixed number for all particles in many practical cases. To address this, we develop an Obi parameter optimisation tool. The developed tool can create new parameter images from reference animation and provide optimal scaling values for the images.

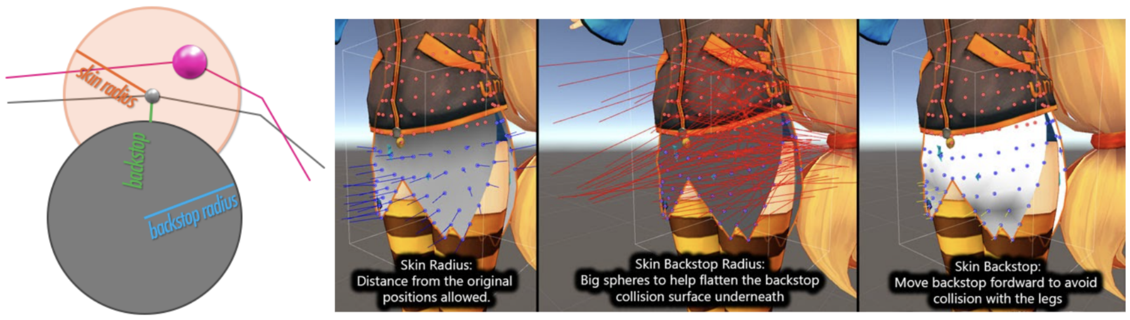

skin radius
backstop
backstop radius
Skin Radius:
Distance from the original positions allowed.
Skin Backstop Radius:
Big spheres to help flatten the backstop collision surface underneath
Skin Backstop:
Move backstop forward to avoid collision with the legs

Figure 18. Particle parameters in Obi-Cloth which can control local garment movements

5.2.1. Obi parameter images

As a first step for understanding the movement of a target garment, we create a reference garment animation from VStitcher. To make this work, we first establish the body-to-garment correspondence using KNN. The heatmap in figure 19 (top) shows the distance from the vertex correspondence. It helps us see how the garment would be placed at the resting position.

If we record the distance over a body animation, we can identify different vertex movements from different garment areas; e.g. a vertex around the chest tends to move less (see figure 19, middle) than the vertex around the thigh (see figure 19, bottom). Based on this observation, we develop a simple algorithm to create the statistics for the garment vertex movement (see figure 20, right).

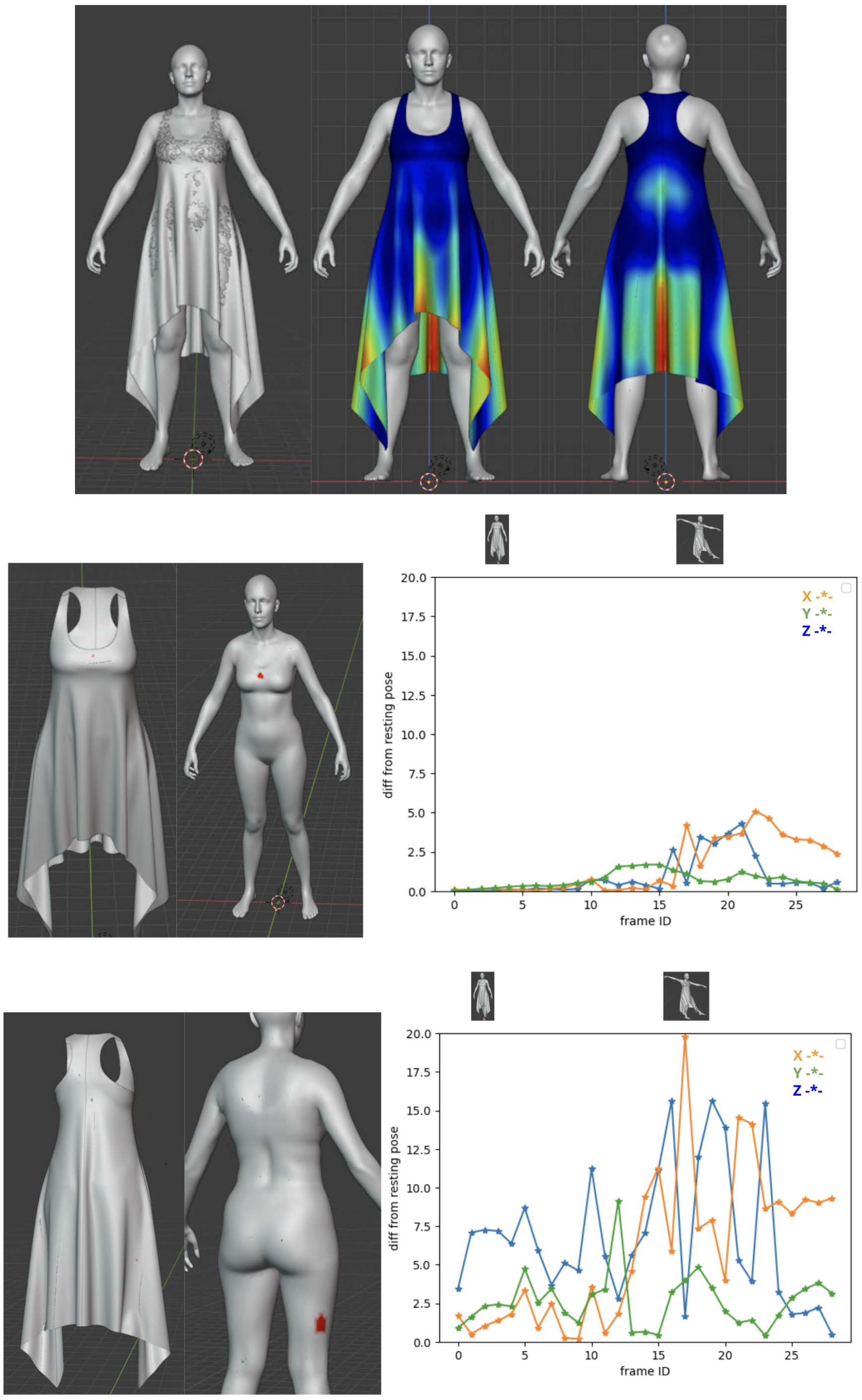
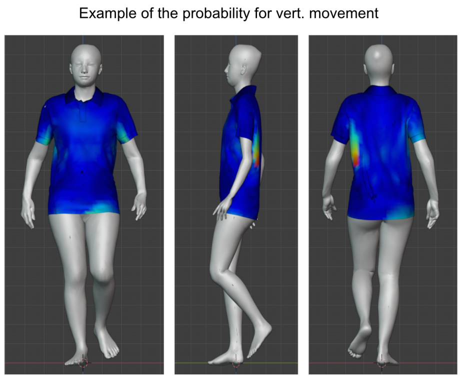


Figure 19. Examples of the vertex movement from a body motion; garment-to-body distance at resting position (top); the vertex movement in the chest area over time (middle); the vertex movement in the thigh area over time (bottom), where each coloured line (orange, green, blue) represents the movement on x, y, z axis, respectively.

The probability for each garment vertex movement is approximated using the standard deviation of the difference between body-to-garment distance at current frame and that from the resting position. We normalise this value with maximum difference, so that the probability is always less than 1.0. This is not a proper probability but it is enough to tell how much each garment vertex would move within the provided motion. We see this information is closely related to the skin radius in Obi-cloth, which defines the maximum radius of particle movement during the simulation.



```
Compute vertex correspondences between body and
garment
Compute initial garment offset at resting pose
for each vertex
For k in all frames
    cur_offset = offset btw body and garment at
    keyframe k
    diff = abs(cur_offset - init_offset)
Compute mean and std for diff(i,:)
prob(i) = std(diff(i,:)) / max(diff(i,:))
```

Figure 20. Examples of the heatmap for garment vertex movement (left); the algorithm creating the vertex movement probability (right)

To convert the per-vertex probability values to a continuous particle parameter image for Obi-cloth, we apply grid interpolation. The resulting values are also smoothed out by a simple blurring filter, which can alleviate the spiky movement during simulation. Figure 21 summarises the overall process. In this example, the front panel of a polo shirt was used. As you can see in the figure, the estimated probability values are discrete values in a 2D UV space (figure 21 left). These discrete values are not useful to Obi-cloth, since it will populate the particles internally at an unknown grid size. Thus, we should make this map continuous so that Obi-cloth will safely retrieve the necessary particle properties. We achieve this with simple image processing techniques and save the result in the red channel of an RGB texture map (see figure 21, middle).

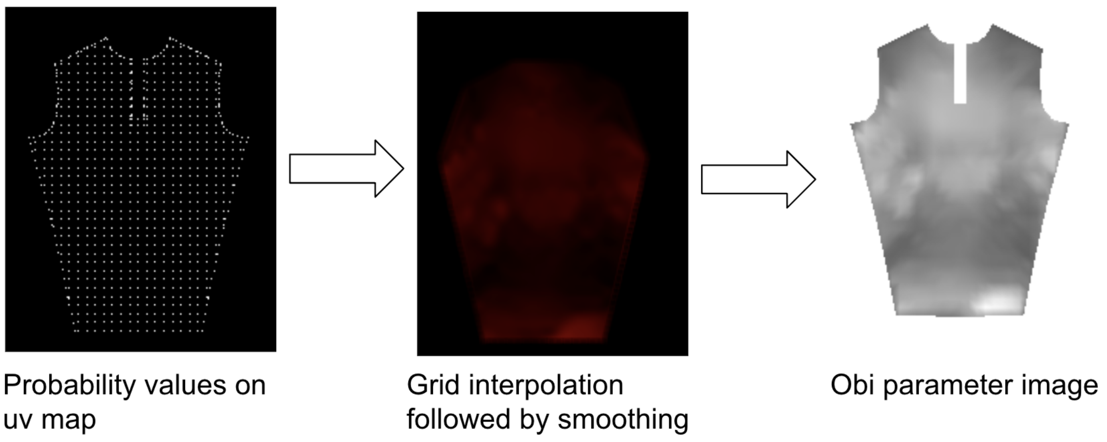

Probability values on uv map
Grid interpolation followed by smoothing
Obi parameter image

Figure 21. Overall conversion process to create an Obi parameter image from the probability values

In addition to the skin radius, we also assume that the backstop skin radius would follow the similar pattern. The backstop skin radius in Obi-cloth works as a limiter of the particle movement. For example, the max range that each particle can move will be defined as *skin radius - (skin radius ∩ skin backstop radius)*. Therefore, if the skin backstop radius is too small, then we can see large garment fluctuations that can go under the body surface. Thus, we assume that the local continuity of the skin backstop radius would be similar to that of skin radius but the scale is higher than the skin radius. Finally, we use a constant backstop value for all particles so that the garment particle is always slightly above the body. At the end, we have the parameter images for skin radius, backstop, and backstop skin radius, but we do not know the exact scale of the map. We see finding this value as an optimisation problem and solve it with 3 cost functions.

5.2.2. Obi parameter optimisation

Parameter images produced in the previous section are not usable without proper scales; without scale, it is just 8-bit values between 0 and 255, which does not have any meaning in Obi physics. We initially saw that finding 3 scaling values is an optimisation problem, and thought this could be easily achievable with standard gradient descent approaches. However, we cannot compute the gradient analytically, because our cost functions are defined on rendering images from Unity (which is a blackbox to the optimiser). Numerical approximation of the gradient is not a good solution either, because it could require rendering multiple times for each small parameter increment, resulting in a long optimisation time.

To address this, we employ a non-gradient optimization technique implemented in the never-grad library (Rapin 2018). This library was originally developed for finding the best hyperparameter values in DL models. We see that is similar to our problem, where we are sweeping three scaling parameters to define the best Obi parmeter images. Internally, the never-grad populates a new sample in a target parameter space in a similar way to the genetic algorithm.

The following three cost functions are developed to solve this optimisation problem, and the computed costs for some sample cases are shown in figure 22.

- Body clipping cost
  We cannot solve the clipping problem of a moving body using single offline optimisation. However, we need the rendering result to be as close as possible. In our implementation body cost function, we count the number of body pixels in a target area (i.e. garment) and compare that with that from the reference image.
- Similarity cost
  This cost measures the pixel value difference between rendering results. Ideally, the simulated result should be identical to the reference image, but it is not possible in many cases. We measure this difference using normalised cross correlation. We also use a RoI box defined from a reference image to penalise if there is any offset in the simulated results
- IoU cost
  The rendering result often translates far from the centre of a body due to strong particle movement. Incorrect skin radius values would also reveal the gap between submeshes, especially when the particle moves too much around the

seam. Intersection over Union (IoU) cost penalises the global translation and small stitching gaps appearing within the garment.

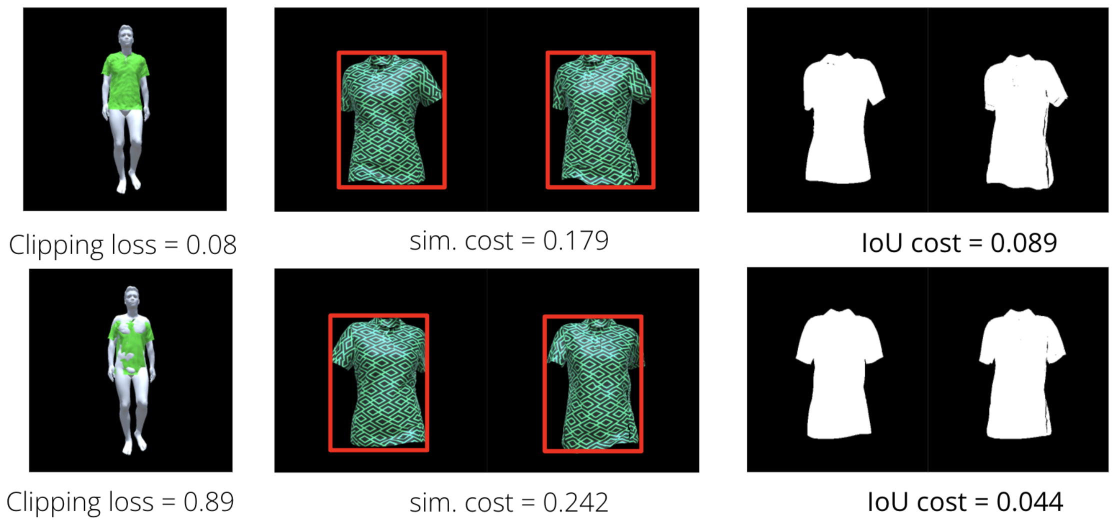


Figure 22. Cost functions used in our Obi parameter optimization and their performance with different samples.

An example of the optimization process is shown in figure 23, which was obtained from the polo shirt shown in figure 20. Obi parmeter images and simple initial guesses for the 3 scales are provided to initialise the optimisation process, and the max iterations is set to 50. In each iteration, our algorithm renders the garment and body using a headless Unity renderer. In this rendering, a predefined test motion was played with and without a body model. This creates a list of images which are compared with the reference images generated from VSticher. Since we sample the motion at every 3-keyframe, the computed costs are averaged.

As shown in figure 23 right, our optimisation samples a parameter space adaptively; e.g. in the early stage it samples sparsely (see iteration id 25) and then when it closes to a local minima (see iteration id from 25 to 42) it starts to sample more densely. We can see the effect of this sampling on the cost space (figure 23, left).

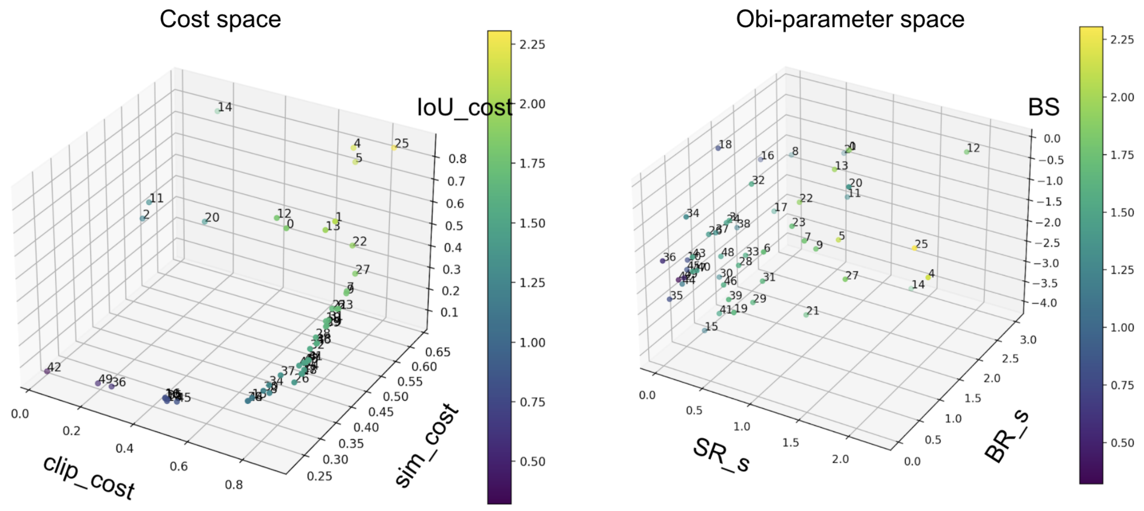
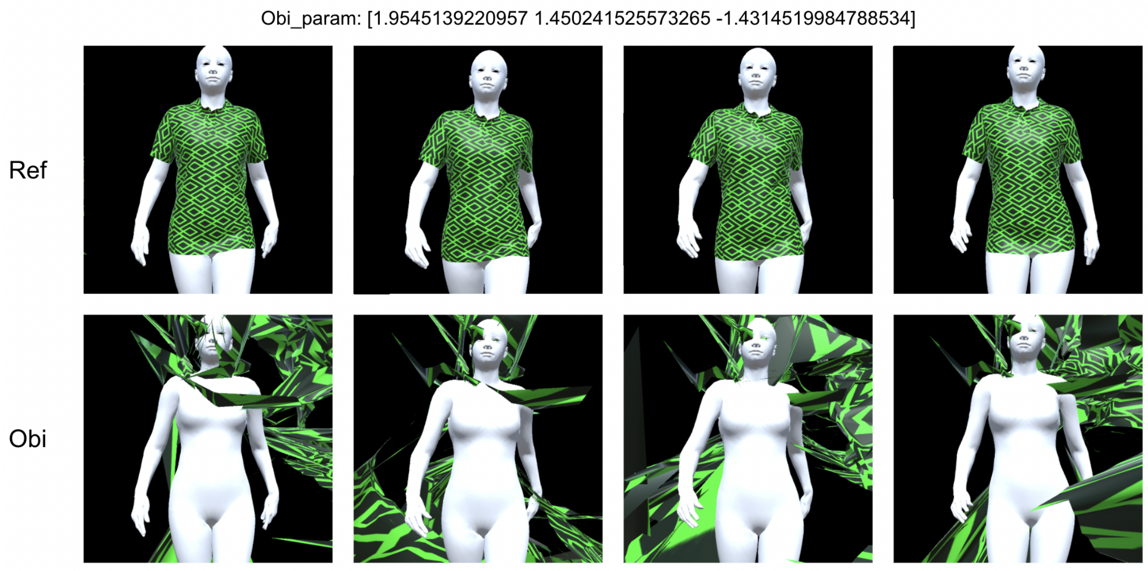


Figure 23. An example of optimisation process: estimated costs over each iteration (left) and the parameter values used in each iteration (right)

The rendering results at iteration 25 and iteration 42 are shown in fig. 5.9 top and bottom, respectively. At iteration 25, our optimiser tests the sample at a distance from the local minimum. Since this sample has a large skin radius (i.e. ~1.95), the garment rendering diverged away from the body. On the other hand, when it hits near to the local minimum, the rendering results appear to be much similar to the reference image from PBS (see figure 24, bottom).


Obi_param: [1.9545139220957 1.450241525573265 -1.4314519984788534]
Ref
Obi

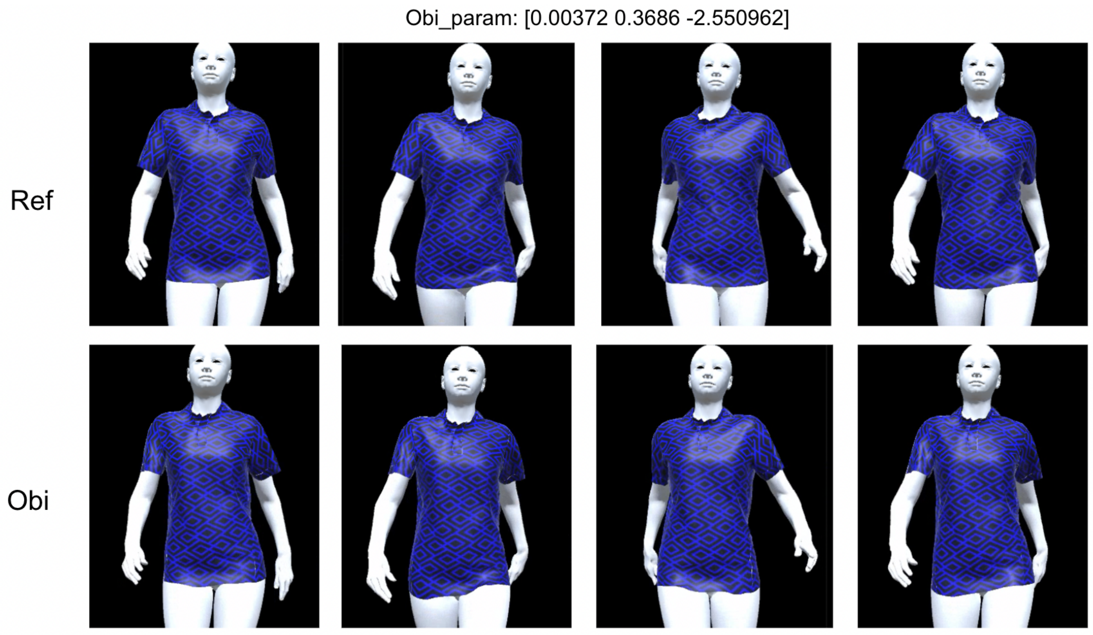


Figure 24. Result from Obi parameter optimisation: the worst result in the early iteration (top, highlighted in green), the final optimisation result (bottom, highlighted in blue)

The overall data pipeline for the Obi parameter optimisation is shown in figure 25. At the moment, the proposed parameter optimisation requires some manual garment data preparation steps using VStitcher. We also found that some Unity assets for a testing garment need to be prepared manually in the headless Unity renderer. This pipeline was recently modified to reuse the same garment skinning process developed for a snap chat lens filter.

The animation baking pipeline explained in Sect 3.1 can be used offline to prepare the animated body models for creating reference garment animations (see figure 25, top left). Obi parameter image creation requires two 3D inputs in different formats; a reference animation in ABC and a UV mapping information from a static FBX. Once the parameter images are ready, Obi parameter optimization will iteratively call the Unity headless renderer to compute the costs. As a result of this process, we can produce multiple parameter images in a UV space and optimised scale values, which will be specific to a target garment and body.

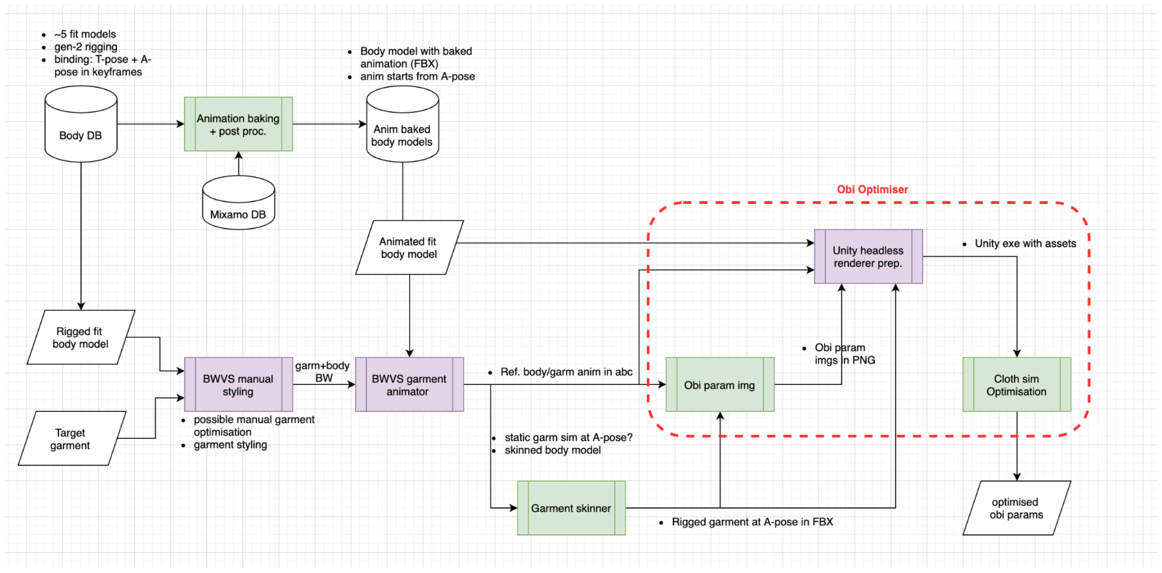


Figure 25. Overall data pipeline for Obi parameter optimisation process, where manual processes are highlighted in purple.

5.3 Body tracking in the Unity application - Skeletal differences in ARKit

At the time of writing Unity only supports body tracking on iOS devices, using Apple ARKit. This is not very well documented by Unity, but we have Apple documentation as reference: Rigging a model for motion capture.

5.3.1 Apple ARKit

ARKit requires a particular skeleton for it to work, and the character needs to be rigged in a T-pose. Apple provides a sample application for testing, and a rigged character. Unfortunately, the character is a robot, so it's not the best representation of a person. The figure below shows the robot with its rig. It has many bones along the spine and neck. Also, you can see the different skinning areas in different colors in the right figure. As you can see, the segments are separate, whereas in a human there should be some smooth transition between the different segments of a limb.

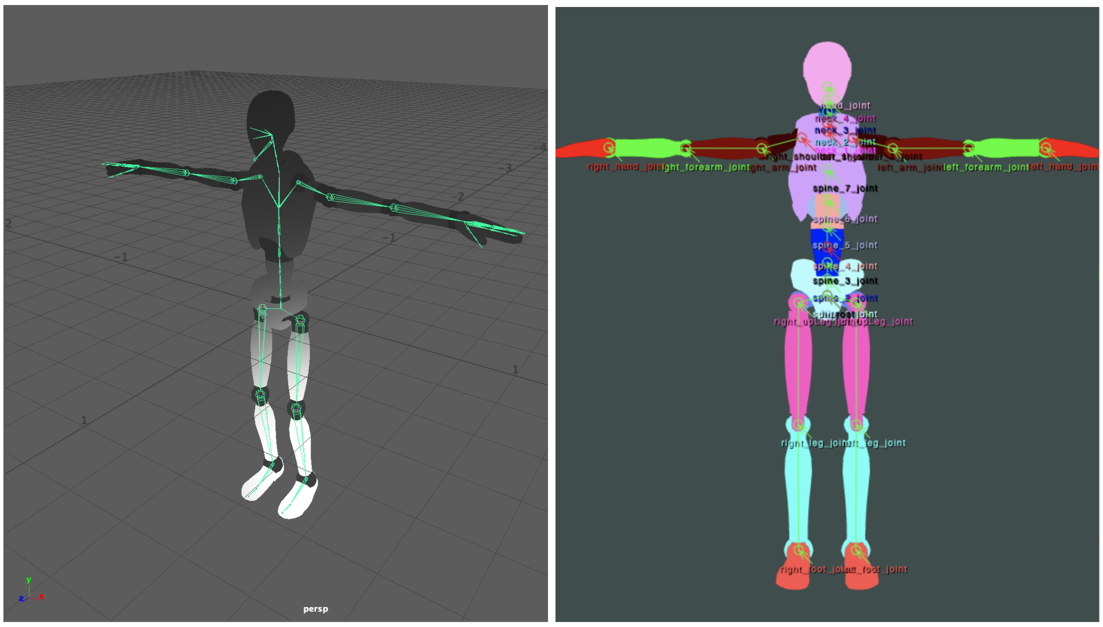


Figure 26. Sample character provided in Apple's Body Tracking sample

We require a different skeleton in order to run the animations needed for the training of our physics parameters, as described in the cloth simulation section. Because of that, we decided to develop a CLI to convert the skeleton of the avatars to an ARKit-compatible avatar in the last step of the process. This is similar to the process described in the Lens Studio section.

The CLI takes an avatar and a script file as an input, and produces a new avatar with the modifications described in the script file. We call that script file the "tweaks file", because it contains a Domain Specific Language that describes a series of "tweaks" for the model and its skeleton. The main operations or tweaks are:

- origin <JOINT>: recenters the model around the given joint. This is necessary because the ARKit model is centered around the hip, whereas our models have their origin between the 2 feet.
- scale <NUMBER>: scales the mesh and skeleton by the given number. The units of our model are in centimeters, whereas for ARKit we need the units to be in meters.
- mv-bone <OLD> <NEW>: renames a bone/joint.
- rm-tree <JOINTS>: removes the given joints and all their children. Their skinning weights will be re-assigned to its parent.
- rm-bone <JOINTS>: removes the given joints and reparents their children to their grandparents. The skinning weights of the removed joints are reassigned to their parents.
- add-bone <PARENT> <CHILD> <NEW_1> … <NEW_N>: adds N new joints between the given parent and child joints. The joints are placed equidistantly distributed between the parent and child, in the given order, that is, NEW_1 will be the first child of PARENT, and CHILD will become the child of NEW_N. The new joints aren't assigned any skinning weight.
- add-bone <JOINT> <JOINT> <NEW_1> … <NEW_N>: this is the same syntax and logic as above, but when PARENT and CHILD are the same bone, it is interpreted as a terminal joint. This is necessary because Apple ARKit requires the skeleton to be exhaustive, that is, for it to work we need to have exactly all the bones that appear in their rig, even if they become dummy terminal bones.

Our tweak file to convert to Apple ARKit looks like this:

```
# Recenter model around hip
origin hip
# Scale from cm to meters
scale 0.01
# Bone renaming
mv-bone lFoot left_foot_joint
mv-bone rFoot right_foot_joint
mv-bone abdomen spine_1_joint
mv-bone abdomen2 spine_4_joint
# [...]
# Remove all these
rm-tree lPectoral rPectoral
# [...]
# Remove single bones (will be reparented)
rm-bone pelvis
# [...]
# Add extra bones
add-bone chest neck spine_7_joint
add-bone head head nose_joint
[...]
```

With that conversion, the model will load in the Apple body tracking sample app. But it won’t work correctly because the local axis of each joint needs to be adjusted as well. The figure below shows the Apple body tracking sample using one of our avatar after applying those tweaks. Note that in order for the model to work, we have to convert the FBX file that our CLI outputs to USDZ format. This can be done by loading the FBX into Maya, exporting it to USD, and then using Apple Reality Converter to export to USDZ.

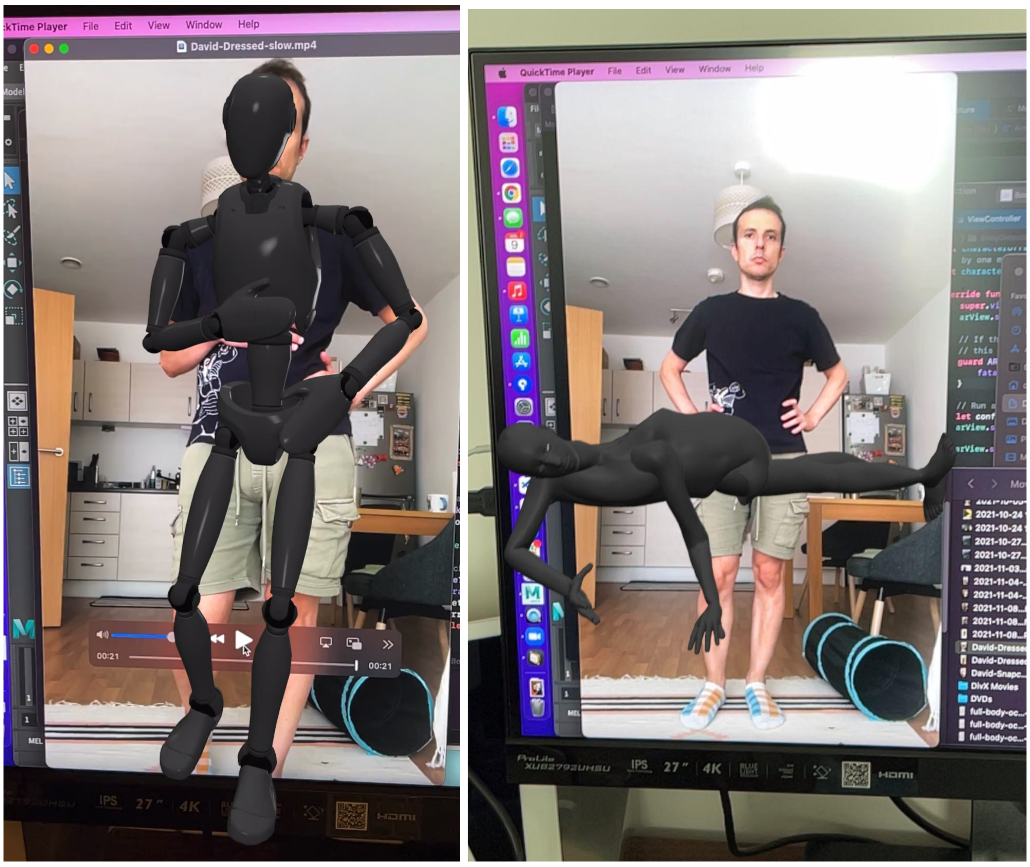


Figure 27. Apple body tracking with the provided character (left) and with our first attempt to import our avatars (right)

In our skeleton, all the joints use a global right-handed axis, where the Y axis points upward, the Z points to the front, and X points to the right. By inspecting the robot FBX in Maya, we can check the necessary rotation for each joint and apply it to our conversion. In general, the Y axis at every joint points forward in ARKit, except for the left side of the body, where it points backward.

The rotations need to be expressed as axis-angle rotations in the exported FBX. By knowing which axis points right and which points up, we can compute the third axis using the cross product. The 3 axes give us an orthonormal base that can be represented as a 3×3 matrix, which can then be converted to an angle-axis rotation. Rotations are accumulated through the hierarchy, so assuming that the first joint of the spine is already facing forward, the next rotation will happen at the shoulder. The figure below illustrates such a transformation for the right shoulder joint.

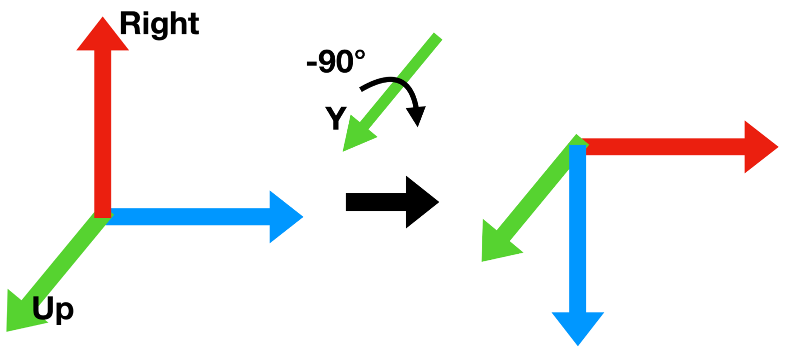


Figure 28. Rotation of spine axis (left) to obtain the right shoulder axis (right)

With all these changes, the avatar now works in the body tracking sample app from Apple. We can also apply the same transforms to the skinned garments. The screenshots below show the ARKit-ready avatar and a polo shirt. Note that the position of the arms seems low, so the positioning of the joints may be too low in our rig, compared to that of the robot.

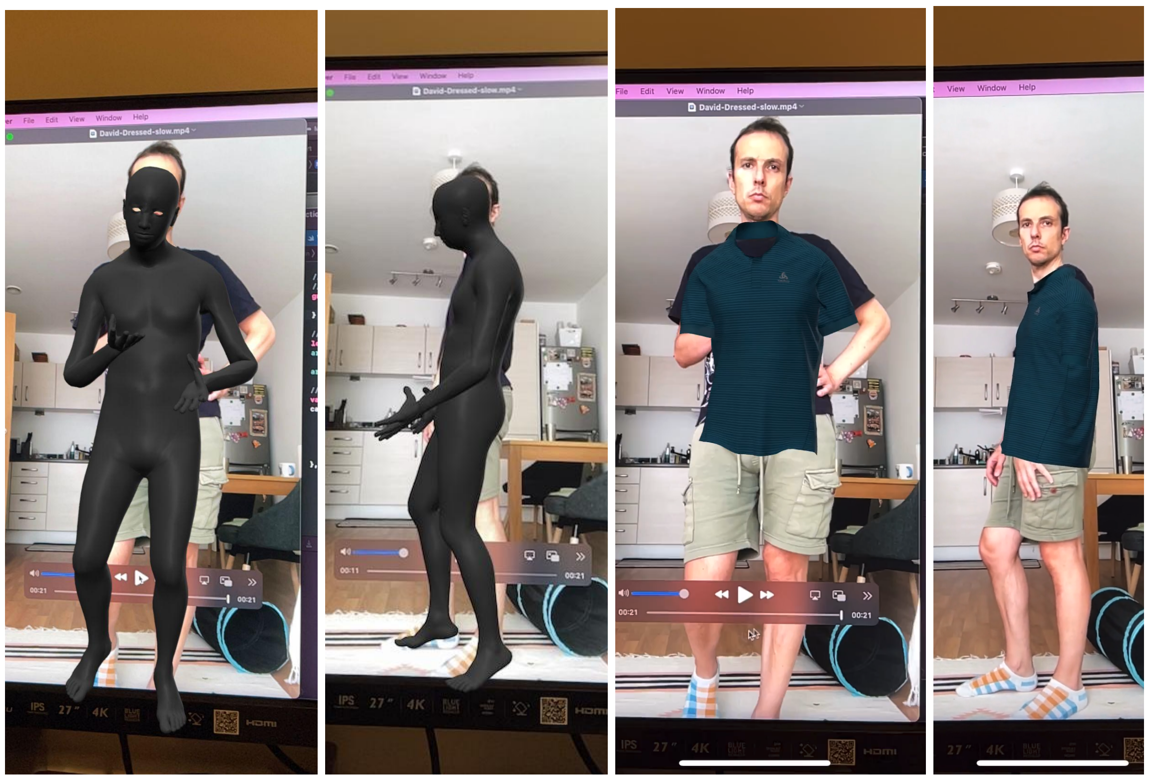


Figure 29. Apple body tracking example with one of our avatars (left) and with one of our garments (right)

5.3.2 Unity ARKit

Unity uses ARKit for body tracking, but the skeleton configuration is slightly different. There is no documentation, but there's an FBX file with the same robot mesh from Apple, but with a slightly different skeleton. The hierarchy of the skeleton is the same, but the names are different, and the rotation axis of each joint is slightly different, usually flipped. There are other small differences, like the starting position of the finger joints. If we simply rename the joints of the skeleton we computed for the ARKit sample app, we obtain a quite funny result in Unity. See below.

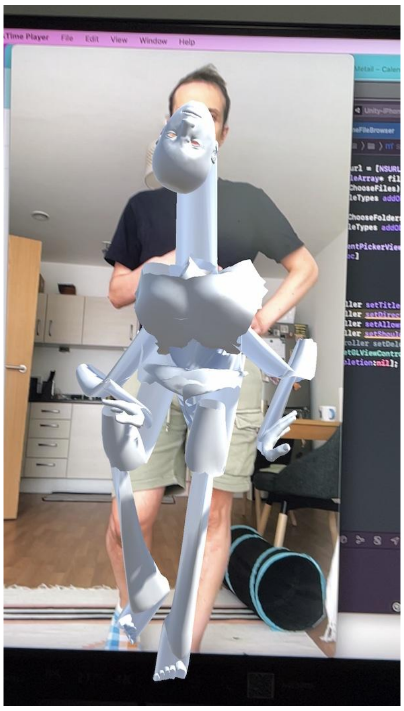


Figure 30. Naive attempt to convert the Apple ARKit avatar model to Unity ARKit.

After inspecting all the axes of the sample robot file and fixing all the axes accordingly, we obtain something closer, as shown in the figure below. However, the forearms look broken for some reason and we haven't discovered why yet. For short-sleeved garments it shouldn't be a problem, though.

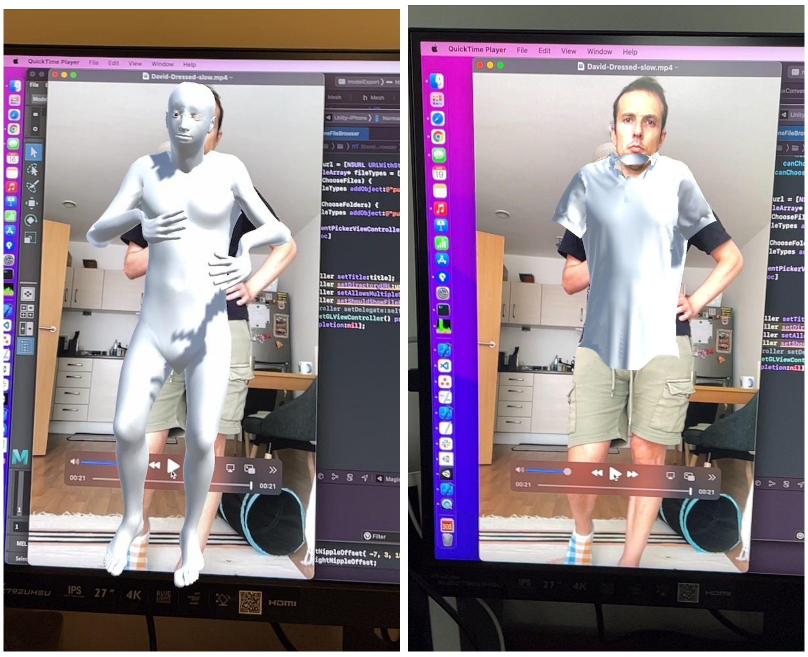


Figure 31. Unity ARKit body tracking example with one of our avatars (left) and with one of our garments (right). Note that there are still issues with the forearm.

6. DressMeUp

6.1 Overview of the application

The DressMeUp application is a consumer application, targeted at social media influencers, but usable by anyone. The concept is simple - the user scans themselves once, then can upload photos of themselves and select digital garments to apply to those photos.

The idea of dressing a photograph with digital garments exists to some extent already, for example on: https://www.instagram.com/thisoutfitdoesnotexist/. However, it is different in two key ways:

1. These are garments which will never exist physically
2. There is a large amount of manual effort in creating these images

The DressMeUp application aims to address both of these points. For the application to add value it needs to:

1. Have better garment representation than a Snapchat Lens.
2. Involve minimal manual processing to compose the final image. Should preferably be completely automated

6.2 Existing Progress and Manual Test

The first main development to achieve a working application was to successfully import the avatar scans from WP1 and transform them into a format usable by our software. This process involved developing an automated pipeline to fix and rig the avatars and has been described in detail in the previous deliverable (D2.1).

To further assess the technical challenges in delivering this project, we created an example output using an Odlo 3D garment while performing various of the required steps manually.

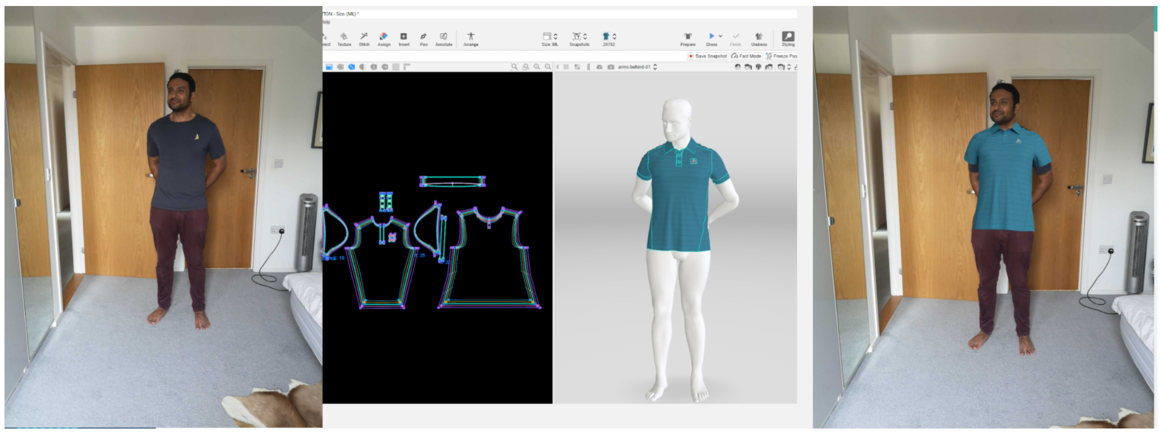


Figure 32. Prototype showing the DressMeUp input and output

Steps in this test:

1. Ran our pipeline to create an avatar suitable for import into VStitcher
2. Manually refined the pose of the avatar to more closely match the photograph
3. Auto-dressed the Odlo 3D garment on to the A-Pose
4. Simulated to the new pose
5. Submitted garment to our rendering pipeline
6. Manually composed the render into the user photo
7. Assess quality of the output

This largely follows the overall flow outlined in the architecture diagrams in WP4, but includes manual steps which we aim to automate in the final output:

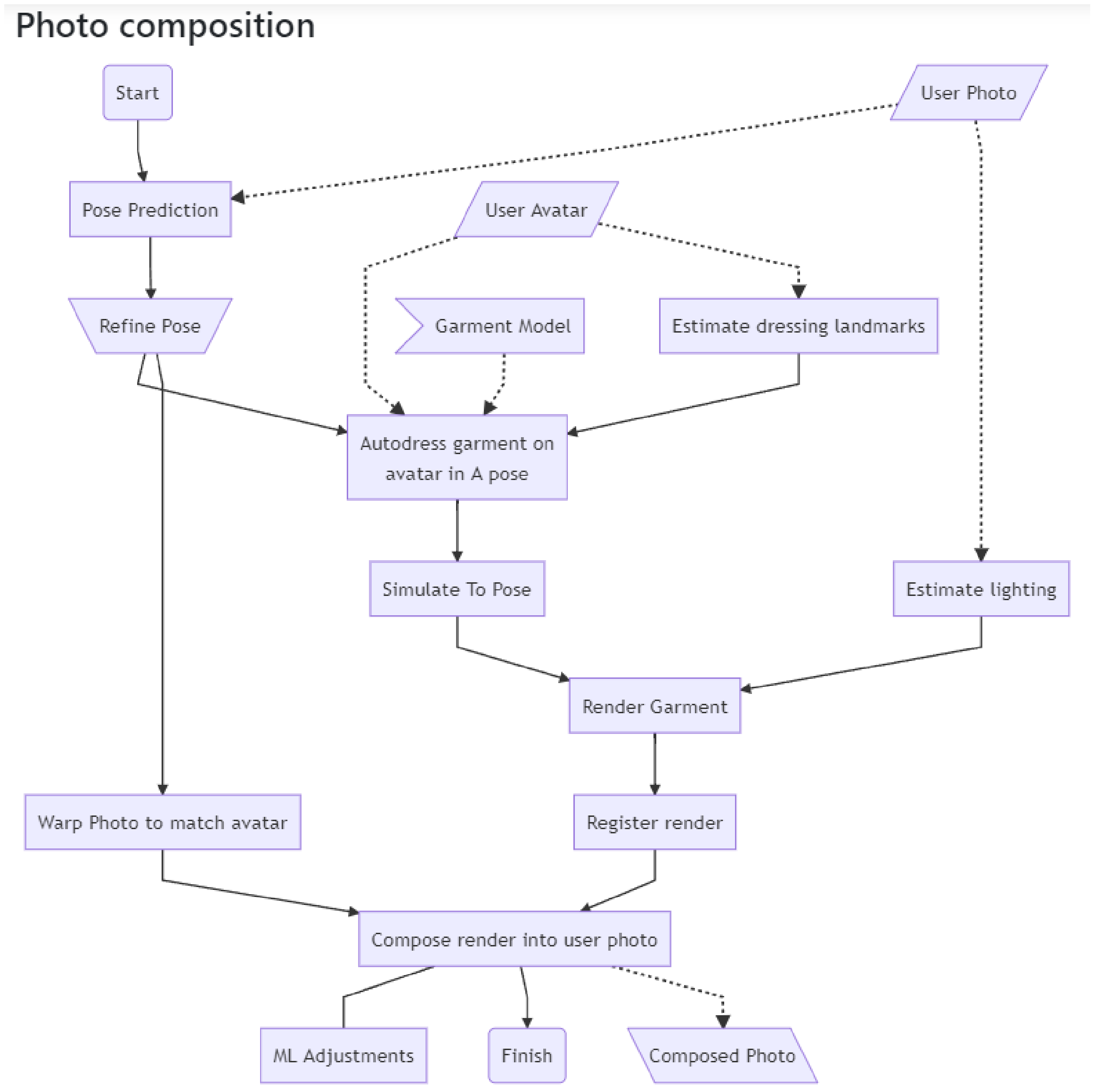


Figure 33. Photo Composition Flow Diagram
https://etryon.gitlab.io/techdocs/arch-diags/use-case-2/

6.3 Evaluating the output of the Manual Test

In general, the output looks of reasonable quality. The garment, though fully digital, looks real and the draping around the body appears realistic. However, if you look closely at the final image, it is clear that there are still some issues with the automated composition of the garment on the body.

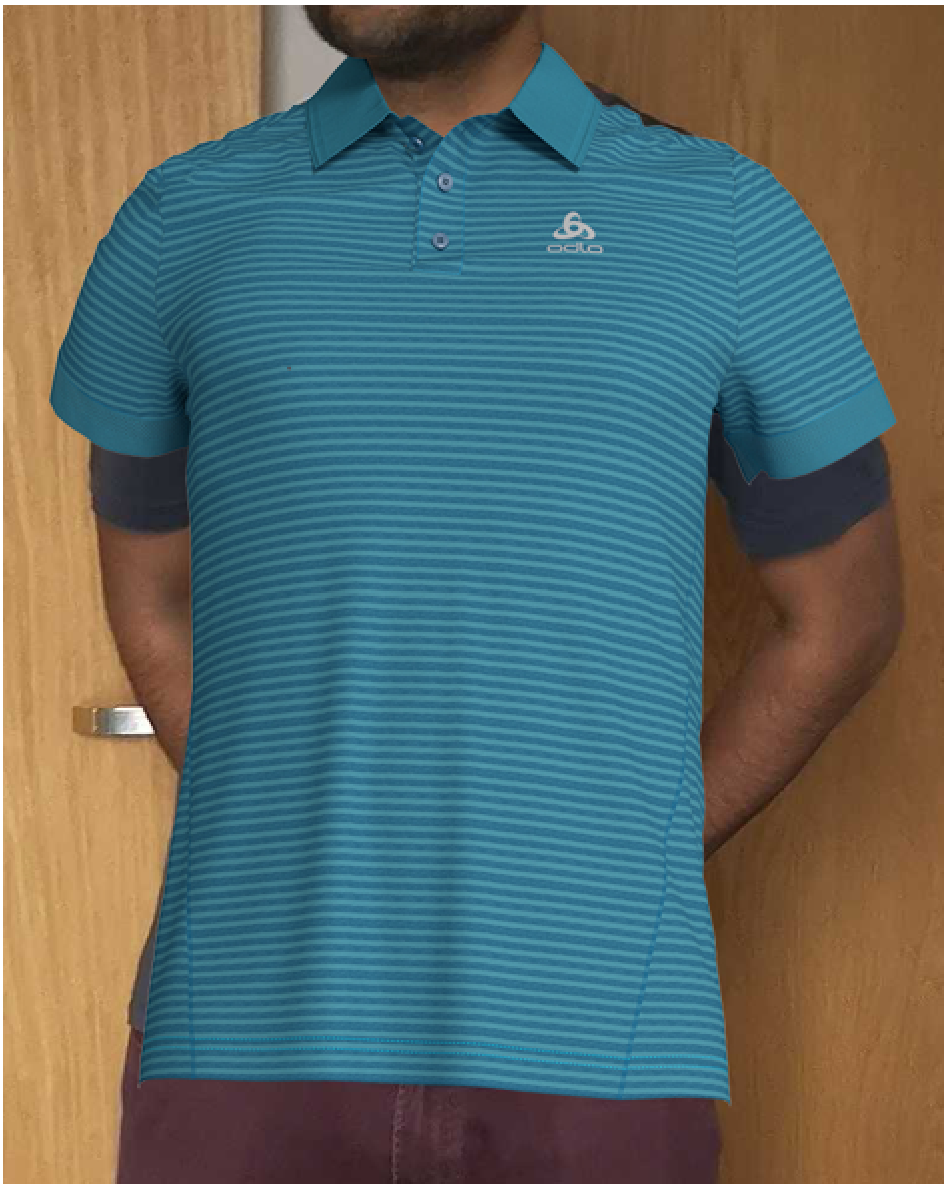


Figure 34. Close up of current prototype output

The main issue is around misalignment between the garment and the body at various points.

In the final application, we need to produce images of sufficient quality for use by influencers. To help us understand which areas to focus on and where the largest misalignments currently occur in our process, we built an evaluation framework.

6.4 Evaluation framework for automatically composed images

To assess the quality of our image compositions, we needed a way to easily generate a multitude of images for review and recording. To do this we created a pipeline which was able to process images in batches and output links to the results in a spreadsheet. These output spreadsheets provide an easy way to collaborate and assess quality in a consistent way.

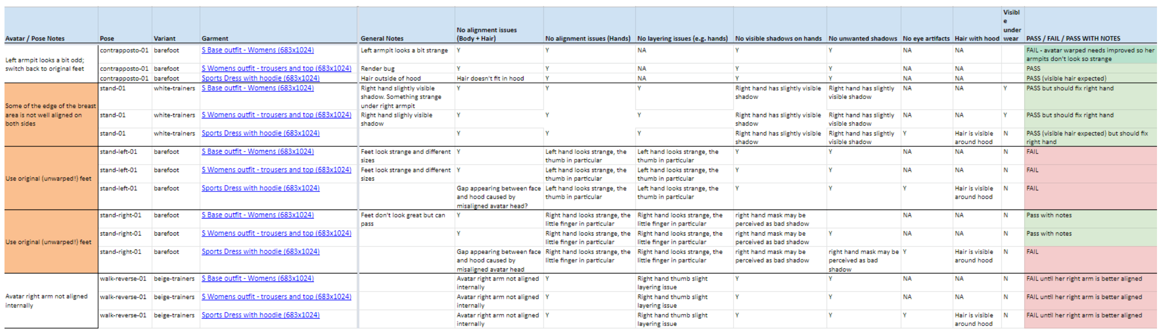
The pipeline generates automated images for a given set of dressed poses and then groups them together into a batch for review. The automated images are produced at two different sizes - maximum resolution and height 1024px. The reasoning behind this is that this will enable us to more easily identify which issues are most problematic. There may be cases where we could offer the user a lower resolution output while the higher resolution image is being corrected.

The completed images are stored in S3 and links to the images are included in the spreadsheet. The sheet then marks columns of common issues which are used as a template to assess the images.

Common issues have been outlined in the spreadsheet:

- No alignment issues (Body + Hair)
- No alignment issues (Hands)
- No layering issues
- No visible shadows on hands
- No unwanted shadows
- No eye artifacts
- Hair visible outside of hooded garment
- Visible undergarments

An example of a completed sheet after assessment can be found here:



Avatar / Pose Notes	Pose	Variant	Garment	General Notes	No alignment issues (Body + Hair)	No alignment issues (Hands)	No layering issues (e.g. hands)	No visible shadows on hands	No unwanted shadows	No eye artifacts	Hair with hood	Visible under wear	PASS / FAIL / PASS WITH NOTES
Left armpit looks a bit odd; switch back to original feet	contrapposto-01	barefoot	S Base outfit - Womens (683x1024)	Left armpit looks a bit strange	Y	Y	NA	Y	Y	NA	NA		FAIL - avatar warped needs improved so her armpits don't look so strange
	contrapposto-01	barefoot	S Womens outfit - trousers and top (683x1024)	Render bug	Y	Y	NA	Y	Y	NA	NA		PASS
	contrapposto-01	barefoot	Sports Dress with hoodie (683x1024)	Hair outside of hood	Hair doesn't fit in hood	Y	NA	Y	Y	NA	NA		PASS (visible hair expected)
Some of the edge of the breast area is not well aligned on both sides	stand-01	white-trainers	S Base outfit - Womens (683x1024)	Right hand slightly visible shadow. Something strange under right armpit	Y	Y	Y	Right hand has slightly visible shadow	Right hand has slightly visible shadow	NA	NA	Y	PASS but should fix right hand
	stand-01	white-trainers	S Womens outfit - trousers and top (683x1024)	Right hand slighly visible shadow	Y	Y	Y	Right hand has slightly visible shadow	Right hand has slightly visible shadow	NA	NA	Y	PASS but should fix right hand
	stand-01	white-trainers	Sports Dress with hoodie (683x1024)		Y	Y	Y	Right hand has slightly visible shadow	Right hand has slightly visible shadow	Y	Hair is visible around hood	N	PASS (visible hair expected) but should fix right hand
Use original (unwarped!) feet	stand-left-01	barefoot	S Base outfit - Womens (683x1024)	Feet look strange and different sizes	Y	Left hand looks strange, the thumb in particular	Left hand looks strange, the thumb in particular	Y	Y	NA	NA	N	FAIL
	stand-left-01	barefoot	S Womens outfit - trousers and top (683x1024)	Feet look strange and different sizes	Y	Left hand looks strange, the thumb in particular	Left hand looks strange, the thumb in particular	Y	Y	NA	NA	N	FAIL
	stand-left-01	barefoot	Sports Dress with hoodie (683x1024)		Gap appearing between face and hood caused by misaligned avatar head?	Left hand looks strange, the thumb in particular	Left hand looks strange, the thumb in particular	Y	Y	Y	Hair is visible around hood	N	FAIL
Use original (unwarped!) feet	stand-right-01	barefoot	S Base outfit - Womens (683x1024)	Feet don't look great but can pass	Y	Right hand looks strange, the little finger in particular	Right hand looks strange, the little finger in particular	right hand mask may be perceived as bad shadow		NA	NA	N	Pass with notes
	stand-right-01	barefoot	S Womens outfit - trousers and top (683x1024)		Y	Right hand looks strange, the little finger in particular	Right hand looks strange, the little finger in particular	right hand mask may be perceived as bad shadow	Y	NA	NA	N	Pass with notes
	stand-right-01	barefoot	Sports Dress with hoodie (683x1024)		Gap appearing between face and hood caused by misaligned avatar head	Right hand looks strange, the little finger in particular	Right hand looks strange, the little finger in particular	right hand mask may be perceived as bad shadow	right hand mask may be perceived as bad shadow	Y	Hair is visible around hood	N	FAIL
Avatar right arm not aligned internally	walk-reverse-01	beige-trainers	S Base outfit - Womens (683x1024)		Avatar right arm not aligned internally	Y	Right hand thumb slight layering issue	Y	Y	NA	NA	N	FAIL until her right arm is better aligned
	walk-reverse-01	beige-trainers	S Womens outfit - trousers and top (683x1024)		Avatar right arm not aligned internally	Y	Right hand thumb slight layering issue	Y	Y	NA	NA	N	FAIL until her right arm is better aligned
	walk-reverse-01	beige-trainers	Sports Dress with hoodie (683x1024)		Avatar right arm not aligned internally	Y	Right hand thumb slight layering issue	Y	Y	Y	Hair is visible around hood	N	FAIL until her right arm is better aligned

Figure 35. Completed Assessment Sheet
https://docs.google.com/spreadsheets/d/1JxozZSpO8CPZsbQ3LAoxOrVQoJvtX6lmeJdu693lUkg/edit?usp=sharing

This now gives us the ability to test, iterate and improve the output as we work on automation.

6.5 Next steps - automatic pose detection and warping

In the project we have now completed a proof of concept which shows that we can make a functioning end-to-end product (albeit with manual steps) and also created a framework to evaluate the quality of the automated output produced. This will enable us to target areas for improvement in our automated image generation.

Next we will be looking at ways to improve the output quality of the automated image generation. The two main themes of research will be:

1. Automating pose detection from a photograph - so that the scanned avatars can be re-posed for dressing in VStitcher without manual effort in re-posing
2. Image warping techniques to improve the alignment between the body and the garment - without distorting the overall look of the output image

Progress on these threads of work will be described in more detail in the next deliverable (D2.3).

7. VR Designer

7.1 Garment simulation in VR Designer application

The VR Designer application is designed as a way for 3D designers to showcase their designs to their product teams in an interactive VR setting. The UI is described in more detail in D5.2 but the key decision in this deliverable is how best to simulate the garment interactions.

Odlo spent a lot of time building on top of the requirements work completed in D6.1, discussing with designers how the VR application can best satisfy the requirements of the wider design team. In the end, two key decisions were reached to showcase the garments in the best way:

1. 3D fit avatars need to be used in the application (rather than scans of the fit models). This is necessary to ensure consistency and dressability. Because the garments are designed on these fit avatars, the garments will fit exactly as intended and there will be no high frequency 'bumps' which will affect how the garment looks
2. The garment simulations will be exported directly from VStitcher to ensure that the VR application shows the garment physics as realistically as possible. Unlike the MagicMirror which requires freedom of movement, the animations for this application are preset. This means we can directly use the cloth simulation generated by VStitcher

These decisions mean that for this deliverable, the work for the VR Designer application becomes firstly about exporting the animation files correctly (as alembic files which can be used in VR) and secondly about working on the avatars to make them as close to human looking as possible.

7.2 Using Browzwear's new 'realistic avatars' to export animation

Browzwear has created two new avatars which look more realistic than typical 3D avatars.

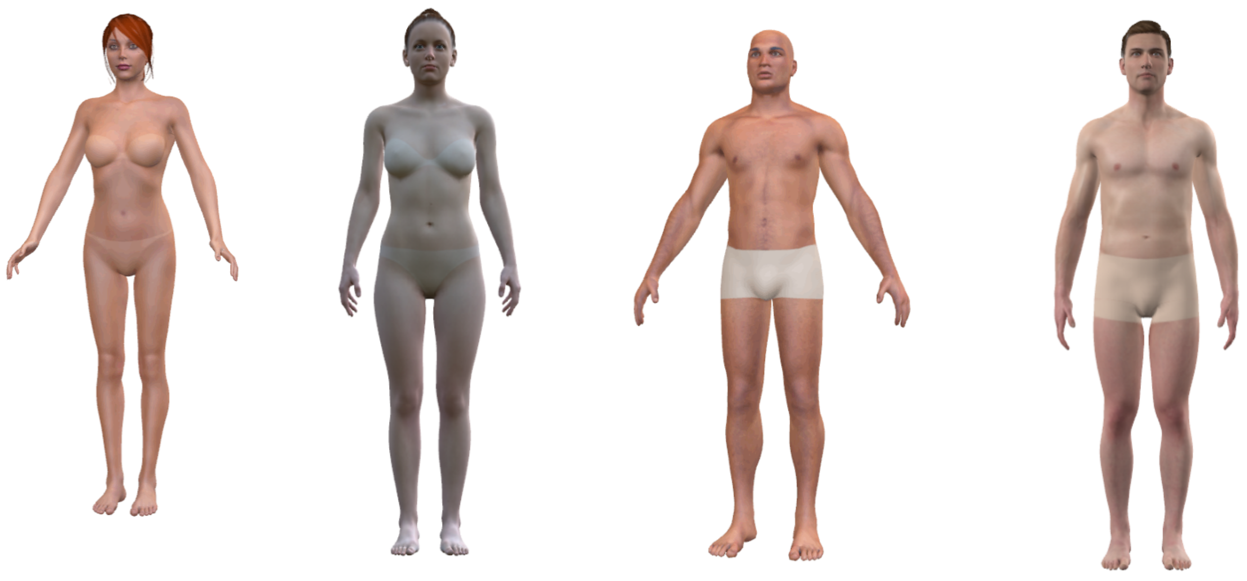
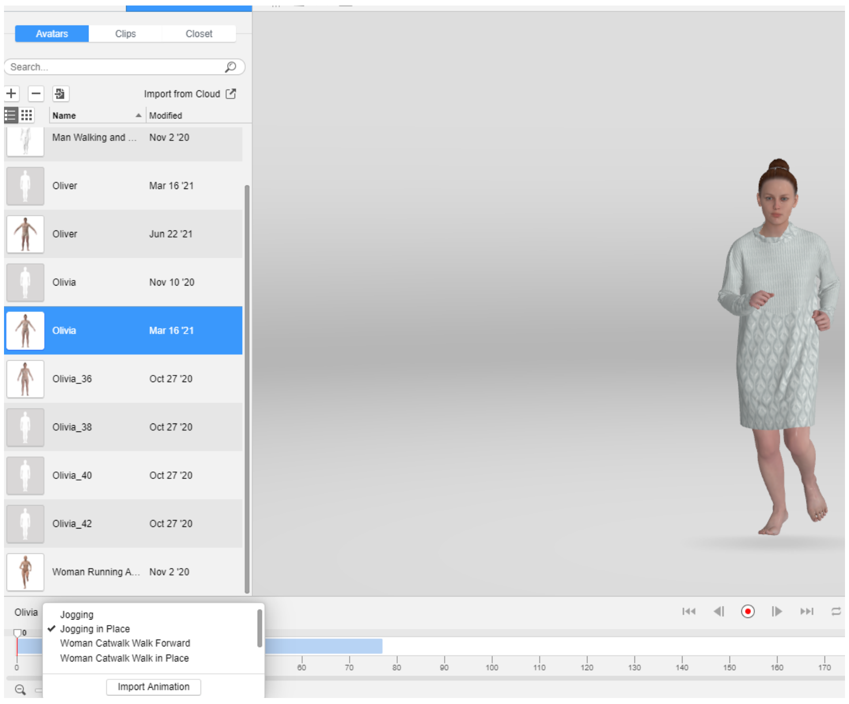


Figure 36. Browzwear's parametric avatars. Previous generation, left, vs current generation, right)

In this figure you can see the previous generation of avatars alongside the latest generation equivalents. Although not yet photorealistic, there is improved realism in both the bodies and the hair.

The other advantage of these avatars is that they are parametric, which means their shape can be changed, and animatable.

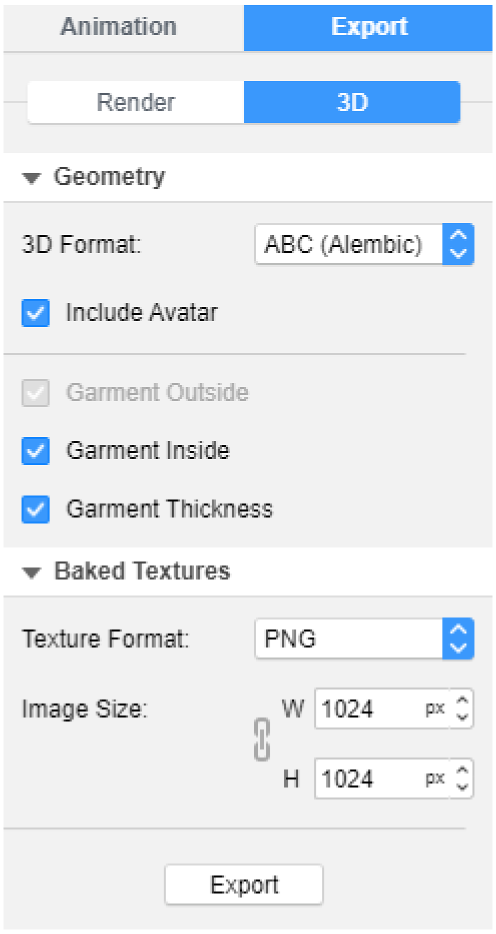


Name	Modified
Man Walking and ...	Nov 2 '20
Oliver	Mar 16 '21
Oliver	Jun 22 '21
Olivia	Nov 10 '20
Olivia	Mar 16 '21
Olivia_36	Oct 27 '20
Olivia_38	Oct 27 '20
Olivia_40	Oct 27 '20
Olivia_42	Oct 27 '20
Woman Running A...	Nov 2 '20

Figure 37. VStitcher's animation workspace

In this figure you can see some default animations that VStitcher has, and also that you can import animations from external services such as Mixamo.

For the VR Designer application, Odlo will be choosing suitable animations to showcase their garments, running the simulations and then exporting them for use in VR.



3D Format: ABC (Alembic)

☑ Include Avatar

☐ Garment Outside

☑ Garment Inside

☑ Garment Thickness

Texture Format: PNG

Image Size: W 1024 px H 1024 px

Figure 38. Exporting alembic files from VStitcher

Odlo are in the process of getting the new parametric avatars created for all of their size sets and will then process the files for import in the application. Further details of this progress will be discussed in the next deliverable D2.3.

8. Conclusion

The objective of this deliverable was to provide the basic working version of avatar-garment simulation software. This document describes in detail the steps taken to successfully reach that point. The effort required for each use-case was different, but we have shown that we can deliver the requirements for each effectively, whether that is through:

- Utilising the latest features in VStitcher (new avatars and animations) and pulling them in to VR *[VR Designer App]*
- Manual end-to-end pipeline plus a criteria for evaluation, with a view to target automation effectively *[DressMeUp App]*
- Building on top of existing technologies and integrating with our systems to create AR lenses *[SnapChat Filter, MagicMirror App]*
- Creating innovative approaches to get to true-to-life cloth simulation on real body shapes in Unity *[Magic Mirror App]*

Creating these first versions of working software show that the research is not only innovative, but also applicable to real world scenarios.

These use cases highlight the significant progress made in the “simulation algorithms for avatar-garment interactions” innovation objectives. We can now visualise professionally designed virtual garments on real people, in both static images and real-time video. We have shown how animations can be taken into VR to share designs in an entirely new way. Finally, we have created a way to predict realistic physics of garments that are made for production by fashion designers in a game engine (Unity), without disrupting the fashion design workflow.

We will be continuing to improve in both quality and automation for these applications as we work towards the next deliverable for the final version of the simulation software.

We believe we can improve the output of this objective by adding a visualisation of how garments fit on a given body. We intend to utilise our learnings in VStitcher so far to create a dataset of animations consisting of different sizes of garments on different sizes of avatars. From this dataset, we aim to replace the garment deformation model with a new deep learning model which will allow garment simulation for different poses and body shapes. If this works as intended, we can then build a machine learning ML model for fit prediction given an arbitrary body input.

9. References

Demetri Terzopoulos, John Platt, Alan Barr, Kurt Fleischer “Elastically deformable models,” SIGGRAPH proc. 87

Kwang-Jin Choi, Hyeong-Seok Ko. “Stable but responsive cloth,” SIGGRAPH02

Miles Macklin, Matthias Müller, Nuttapong Chentanez, Tae-Yong Kim “Unified particle physics for real-time applications,” SIGGRAPH14

Ben Mildenhall and Pratul P. Srinivasan and Matthew Tancik and Jonathan T. Barron and Ravi Ramamoorthi and Ren Ng “NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis,” ECCV20

Jonathan T Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, Pratul Srinivasan “Mip-NeRF: A Multiscale Representation for Anti-Aliasing Neural Radiance Fields,” ICCV21

Guan, P. and Reiss, L. and Hirshberg, D. and Weiss, A. and Black, M. J. “DRAPE: DRessing Any PErson,” ACM Trans. on Graphics (Proc. SIGGRAPH12)

Erhan Gundogdu, Victor Constantin, Amrollah Seifoddini, Minh Dang, Mathieu Salzmann, Pascal Fua “GarNet: A Two-Stream Network for Fast and Accurate 3D Cloth Draping,” ICCV19

Patel, Chaitanya and Liao, Zhouyingcheng and Pons-Moll, Gerard, “TailorNet: Predicting Clothing in 3D as a Function of Human Pose, Shape and Garment Style,” CVPR20

Saito, Shunsuke and Yang, Jinlong and Ma, Qianli and Black, Michael J. “Scanimate: : Weakly Supervised Learning of Skinned Clothed Avatar Networks,” CVPR21

Qianli Ma, Jinlong Yang, Siyu Tang and Michael J. Black. “The Power of Points for Modeling Humans in Clothing,” ICCV21

Zorah Lahner, Daniel Cremers, Tony Tung “DeepWrinkles: Accurate and Realistic Clothing Modeling,” ECCV18

Ma, Qianli and Yang, Jinlong and Ranjan, Anurag and Pujades, Sergi and Pons-Moll, Gerard and Tang, Siyu and Black, Michael J. “CAPE: Learning to Dress 3D People in Generative Clothing” CVPR20

C. R. Qi, H. Su, K. Mo, and L. J. Guibas “Pointnet: Deep learning on point sets for 3d classification and segmentation,” arXiv preprint arXiv:1612.00593, 2016.

Igor Santesteban, Miguel A. Otaduy, and Dan Casas “Learning-Based Animation of Clothing for Virtual Try-On,” Computer Graphics Forum (Proc. of Eurographics19)

Santesteban, Igor and Thürey, Nils and Otaduy, Miguel A. and Casas, Dan "Self-Supervised Collision Handling via Generative 3D Garment Models for Virtual Try-On," CVPR21

Qianli Ma, Shunsuke Saito, Jinlong Yang, Siyu Tang and Michael J. Black. ”SCALE: Modeling Clothed Humans with a Surface Codec of Articulated Local Elements,” CVPR21

Hugo Bertiche, Meysam Madadi, Sergio Escalera “CLOTH3D: Clothed 3D Humans,” ECCV20

Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” NAACL-HLT (1) 2019

Loper, Matthew and Mahmood, Naureen and Romero, Javier and Pons-Moll, Gerard and Black, Michael J. “SMPL:A Skinned Multi-Person Linear Model,” ACM Trans. Graphics (Proc. SIGGRAPH Asia15)

Anguelov, D., Srinivasan, P., Koller, D., Thrun, S., Rodgers, J., Davis, J.: Scape “shape completion and animation of people,” ACM Trans. Graph. 24, 3 (Proc. Siggraph05)

Mahmood, Naureen and Ghorbani, Nima and Troje, Nikolaus F. and Pons-Moll, Gerard and Black, Michael J. “AMASS: Archive of Motion Capture as Surface Shapes,” ICCV19

J. Rapin and O. Teytaud, “Nevergrad - A gradient-free optimization platform,” GitHub repository 2018, https://GitHub.com/FacebookResearch/Nevergrad