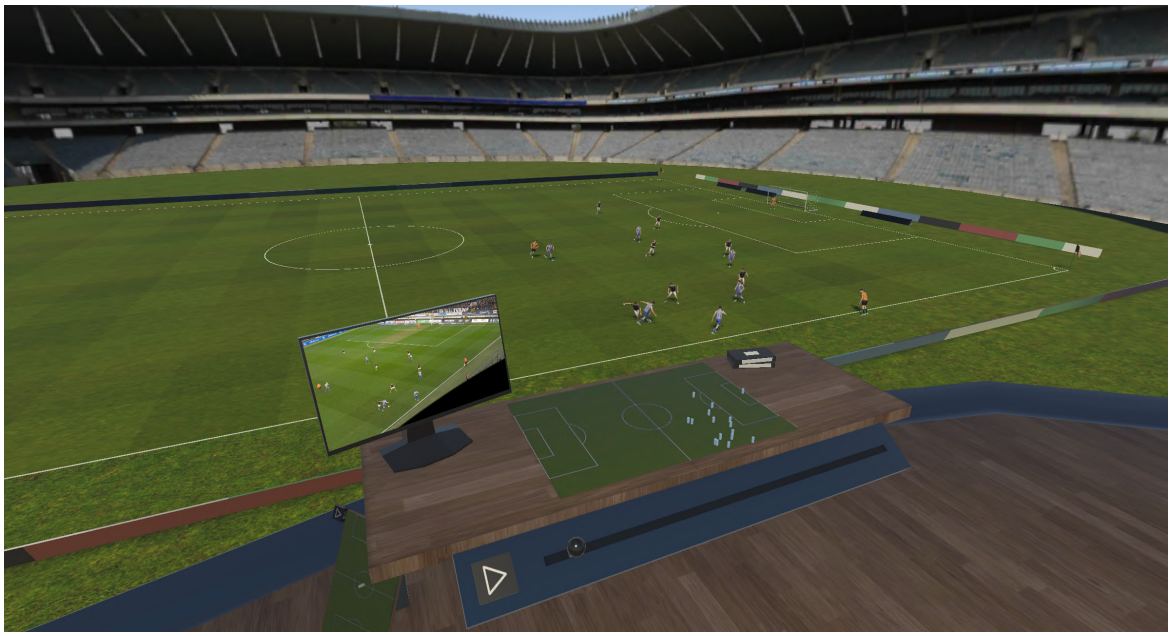


Football Analysis in VR

Texture Estimation with Differentiable Rendering and Diffusion Models

Master's thesis in Data Science and AI

Filip Anjou
Albin Ekström

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2024
www.chalmers.se

MASTER'S THESIS 2024

Football Analysis in VR

Texture Estimation with Differentiable Rendering and Diffusion Models

Filip Anjou
Albin Ekström

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2024

Football Analysis in VR

Filip Anjou
Albin Ekström

Supervisor and Examiner: Lennart Svensson, Department of Electrical Engineering

Master's Thesis 2024
Department of Electrical Engineering
Chalmers University of Technology
SE–412 96 Gothenburg
Telephone +46 31 772 1000

Cover: View of the virtual football pitch from the starting area

Typeset in LaTeX
Gothenburg, Sweden 2022

Abstract

In recent years, football analysis has grown in popularity with various new tools, ranging from tracking entire teams' movement patterns to monitoring the effort exerted by individual players during matches. One increasingly popular method, facilitated by improved cameras, is recording matches and analysing them afterwards. However, some players may struggle to translate the overhead perspective typically used by cameras to their playing style. This master's thesis focuses on using a single camera angle to reconstruct a match sequence in 3D, allowing interaction with the environment in virtual reality (VR) from arbitrary viewpoints.

Together with two other master's theses, a pipeline has been developed where the results from the first project, which tracks each player and the ball over time, are given to the second project, which estimates the poses of all players and produces the results for this master's project. This project concentrates on creating the interactive VR environment. Additionally, a texture generation testbed has been created to automatically recreate the players' clothing in the 3D environment, to make the pipeline from a video to an interactable environment as seamless as possible.

A VR environment was constructed for the visualisation of, and interaction with, three-dimensional reconstructions of recorded football matches. Within this environment, the user may control playback, teleport onto the pitch, and even experience the match from the perspective of one of the players. Furthermore, a texture generation testbed was built utilising differentiable rendering to estimate the football players' kits based on sequences of video frames and their estimated pose. Various techniques were explored, including perceptual loss and the application of diffusion models for inpainting and image-to-image synthesis, both in texture space and the image plane.

The texture generation testbed produces convincing results in best-case scenarios for players visible from many angles with accurate pose estimates. However, the combination of low-resolution images, lack of views for certain players, and even slight misalignments between the estimated pose and the player in the image increases the reliance on synthetic imagery. These methods require further tuning, and possibly training, to produce life-like results.

Acknowledgements

We wish to express our gratitude to our examiner and supervisor, Lennart Svensson, for his invaluable thoughts, perspectives, and advice throughout this thesis. We also extend our thanks to our supervisor, Anders Sjöberg, at Fraunhofer-Chalmers Centre for Industrial Mathematics (FCC) for his support and significant contributions. Additionally, we are grateful to IFK Göteborg for the football videos and their significant opinions and we appreciate FCC's overall support for making this thesis possible. Finally, we would like to extend our gratitude to our fellow Master's thesis colleagues: Hugo Ganelius, Jhanzaib Humayun, Joakim Osterman, and Olof Sjögren.

With sincere thanks,
Filip Anjou
Albin Ekström

Glossary

Diffusion Model (DM)	A generative neural network trained by denoising artificially corrupted images.
Face	A surface defined by three (or more) vertices.
Fragment	A point in screen space containing colour and depth information.
Latent Diffusion Model (LDM)	A diffusion model that is trained on low-dimensional (latent) representations of images to save computational resources.
Mesh	A set of vertices and faces representing an object in computer graphics.
Score Distillation Sampling (SDS)	A technique for training models to generate three-dimensional geometry using a pre-trained diffusion model trained on two-dimensional images.
Skinned Multi-Person Linear (SMPL) Model	A vertex-based parameterisation of human body shape and pose.
Texture	An image controlling the appearance of a mesh.
Texture Element (texel)	The smallest unit of a texture, storing colour, surface geometry, or similar attributes.
UV	Two-dimensional coordinates in texture space, normalised within $[0, 1]$.
Vertex	A three-dimensional point used to define the shape of a mesh.
Virtual Reality (VR)	A first-person experience in a simulated environment, facilitated using a headset and handheld controllers.
Clip	A video sequence from a football match.
Frame	A still frame from a clip.
Patch	A close-up image of a player cropped from a frame.

Contents

List of Figures

List of Tables

1

Introduction

The usage of analysis tools in football has increased a lot in recent years and today almost every club has a camera system installed, even in grassroots football clubs [1, 2]. However, according to M. Edlund, football developer at IFK Göteborg [3], while some players tend to learn from drawing on paper and others by analysing videos of football matches, some require a more realistic experience to understand and internalise the coach's intentions.

Hence, this master's thesis, along with two other theses [4, 5], introduces a novel football analysis tool for interacting with reconstructed match sequences from a first-person perspective. This initiative is a collaboration between the local football club IFK Göteborg, Chalmers University of Technology, and Fraunhofer-Chalmers Centre for Industrial Mathematics. Given a single video clip from a real football match, the system extracts match data — including details about the players and the ball — into a 3D environment for post-match analysis using a virtual reality (VR) headset. We build a demo that replicates real football sequences with all players and match actions, akin to the FIFA video game [6] but based on actual footage captured from a single camera.

The first work [4] attempts to continuously identify and track each player throughout the match. They also focus on pinpointing the ball at all times. The second work [5] estimates each player's pose utilising the tracking data from the first work. By estimating every player's pose in each frame, they can be made into animations which power a realistic 3D reconstruction.

To create a seamless experience with our pipeline, we want the user to submit a video clip and receive a complete VR experience. To achieve this, one essential element is that the reconstructed players must have their kits automatically generated, also referred to as player textures. Several previous works have shown that recreating textures of what people are wearing is feasible with high-resolution footage. Among these works, some utilise Neural Radiance Fields [7], as demonstrated in works such as [8–11]. Alternatively, other methods are based on Gaussian Splatting [12], exemplified by studies including [13, 14]. Both techniques aim to accurately reconstruct 3D scenes and objects, including both geometry and appearance, from 2D images. Then there is the third category, which directly estimates textures from footage using diffusion models [15] as shown in works like [16] and [17].

Given that most cameras[1] used to record football matches are fixed in position and aimed to

[1]This is evidenced by the fact that nearly all football clubs in Sweden [1, 2] use a camera system provided by either Spiideo [18] or Veo [19].

capture the entire football pitch, optical zooming in the footage afterwards is not possible. Instead, digital zoom is used, which degrades the resolution. Consequently, most recorded football matches are limited by their resolution; hence, there is a gap between existing research focused on texture generation (demanding high-resolution quality) and the prevalence of low-quality footage. Considering the already limited resolution of the footage, our objective is to zoom in on individual players for tracking, pose estimation, and texture estimation, which consequently reduces the resolution even further. This means that the image of the player is rarely wider than a few dozen pixels, which presents a fundamental problem.

This master's thesis, the third work, presents an interactive VR environment and a novel method for estimating player textures from low-resolution images mirroring the real match footage, creating a visually appealing and in-depth demo. The interactive VR environment is built in a game engine from the output of the first and second work. Additionally, a differentiable rendering pipeline is built to serve as a testbed for exploring different algorithms to produce player textures.

1.1 Thesis Objectives

This thesis aims to reconstruct a football match in a VR environment together with associated player textures from estimated 3D models. As mentioned, this thesis is based on two other works, which provide us with data to create the environment and generate the player textures.

More specifically, this thesis project aims to:

- Develop a system that accurately produces realistic, dynamic 3D reconstructions of football matches based on sequences of 3D model pose animations. The system will be integrated into an interactive and immersive VR environment, enabling users to experience the match as if they were physically present on the football pitch. Users will have the ability to move around and view the match from various angles and players' perspectives.
- Reconstruct high-resolution texture maps from low-resolution images to produce accurate and visually appealing player textures, utilising state-of-the-art differentiable rendering techniques.

1.2 Limitations

A project of this magnitude involves countless possibilities. Thus, limiting the work is necessary. This thesis does not focus on:

- Expanding the scope of interactive features beyond basic user movement around the pitch. The focus of this thesis is on enabling interactive navigation within a 3D environment and providing a pleasant visual experience for the user.
- Replicating intricate details such as text or sponsor logos on team uniforms. Instead, it will concentrate on accurately reconstructing the primary colours and patterns.

1.3 Contributions

This thesis project has contributed with the following:

- This work showcases the methodology and process for **building a comprehensive VR environment** that realistically reconstructs a **football game from a single camera perspective**. This achievement is integrated with insights and data from two other thesis projects.
- The thesis demonstrates, both theoretically and through experiments, how **textures** can partially be **reconstructed from low-resolution images** of players. This was achieved through **differentiable rendering**, where this thesis has explored various approaches including preprocessing by extracting colour palettes, altering the learning process through different cost functions, and post-processing using methods such as diffusion models.

1.4 Thesis Outline

In Chapter 2, we provide the reader with the necessary background to comprehend the rest of the thesis. We introduce concepts such as differentiable rendering, the Skinned Multi-Person Linear (SMPL) model, and diffusion models.

In Chapter 3, we give an overview of the entire system, including the VR pipeline and the differentiable render model. The primary objective of this chapter is to present the various components of the thesis in a clear and comprehensible manner, facilitating a better understanding of the details discussed later.

In Chapter 4, we describe the VR environment for football match reconstruction. This chapter offers insights into the input data from the other theses and explains how the 3D environment is constructed. We also discuss how it is optimised for VR performance and how interactivity is added.

In Chapter 5, we provide a detailed description of the texture generation model used for estimating player textures with the aid of differentiable rendering. This chapter is structured unconventionally, beginning with an explanation of the raw and fundamental model's construction and functionality. We then analyse the raw results, identify the issues with the estimated textures, explore the underlying causes of these issues, and finally, propose and implement solutions to address them.

In Chapter 6, we conclude the thesis by summarising our results and providing a short discussion.

In Chapter 7, we provide further research areas and potential improvements to how the textures could be estimated even better.

2

Background

This chapter introduces and explains the key concepts that are core to this thesis. These include techniques from computer graphics, such as rasterisation, texture mapping, and differentiable rendering. The Skinned Multi-Person Linear (SMPL) model is also described: a parameterised model for representing humans of various body shapes in arbitrary poses. Finally, an overview of denoising diffusion models is presented.

2.1 Computer Graphics

In computer graphics, the virtual environment, or *scene*, typically consists of a set of objects, one or more sources of light, as well as a virtual camera for capturing the scene. The geometry of an object is often represented by a *mesh*. A mesh consists of a set V of *vertices* $\mathbf{v} \in \mathbb{R}^3$, which are points in 3D space that define the shape of the object. Triplets of vertices can be interconnected to form triangular *faces* $\mathbf{f} = (\mathbf{v}_i, \mathbf{v}_j, \mathbf{v}_k),\ \mathbf{v}_i, \mathbf{v}_j, \mathbf{v}_k \in V$ which define the surface of the mesh (see Fig. 1). Each face has a *normal* direction, which is a vector orthogonal to its surface. Among other things, the normals are used to determine how light is reflected by the object's surface. Since any face can have two possible normal vectors pointing in opposite directions, the chosen vector is often determined by the *winding* of the vertices composing the face. This winding is defined by the order of the vertices in the computer's memory and is combined with a right-hand rule. The normal vector points out of the surface when the vertices are in clockwise order, and into the face when counter-clockwise (or vice versa).

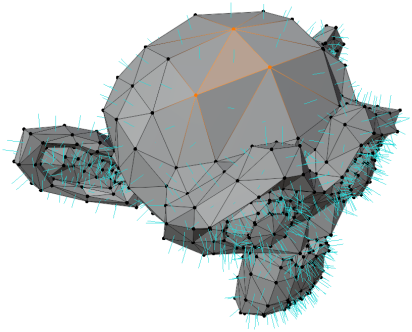


Fig. 1. A mesh with a triangular face highlighted. Black (and orange) dots are vertices. Cyan line segments are surface normals.

2.1.1 Rasterisation-Based Rendering Pipeline

The purpose of a rendering pipeline is to paint the contents of the scene onto the screen's pixels, as viewed from the virtual camera. There are numerous ways of approaching this, including rasterisation, ray tracing, and path tracing. The choice of technique is a compromise between the speed of rendering and the quality of the rendered image; path tracing can produce photorealistic results but is computationally expensive, while rasterisation is fast at the expense of relying on approximation, particularly for lighting calculations such as shadows and reflections.

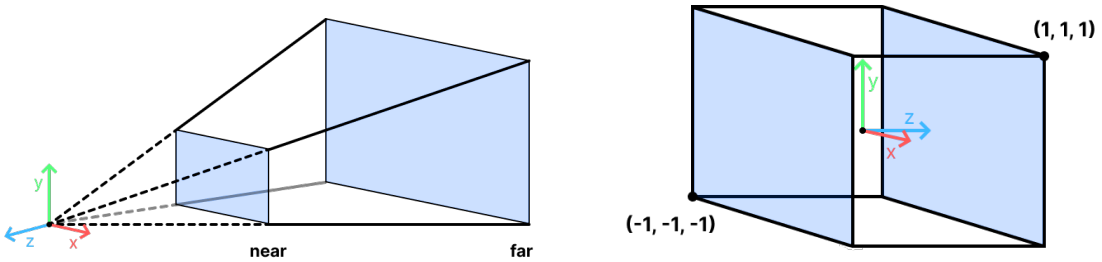
A rendering pipeline based on rasterisation contains a sequence of steps including the transformation of scene geometry from world space to camera space, projection onto the screen, and rasterisation into pixels [20]. These transformations are applied to each triangle individually, making the process trivial to perform in parallel. While the exact steps and coordinate systems used varies between frameworks, the ones outlined below constitute a representative summary.

The first step is the transformation from world coordinates into camera coordinates using a *view matrix* M_V. This transformation is based on the extrinsic parameters (the position $\mathbf{t}$, expressed as a translation matrix T, and rotation R) of the camera in world coordinates. Conceptually, the scene geometry, along with the camera itself, is transformed such that the camera is positioned at the origin, oriented to point in the $-z$ direction with the $+x$ direction to its right. A vertex $\mathbf{v}_w$ in homogeneous world coordinates is thus transformed into $\mathbf{v}_c$ in camera space:

$$\mathbf{v}_c = M_V \mathbf{v}_w, \tag{1}$$

where the view matrix $M_V = (RT)^{-1}$ consist of a rotation and translation.

The camera's view frustum is a six-sided truncated pyramid defined by the field of view of the camera, the aspect ratio of the image, and the near n and far f plane, as shown in Fig. 2a. The next step is a perspective projection that transforms the frustum to a *canonical view volume* in the form of a cube with side length 2, centred at the origin, as shown in Fig. 2b.



(a) Virtual camera's view frustum.

(b) Canonical space.

Fig. 2. Perspective projection of view frustum to canonical space.

This is achieved using a projection matrix

$$M_P = \begin{bmatrix} \frac{1}{a \tan\left(\frac{\theta_v}{2}\right)} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan\left(\frac{\theta_v}{2}\right)} & 0 & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}, \tag{2}$$

where θ_v is the vertical field of view of the camera, $a = \frac{W}{H}$ is the aspect ratio of the rendered image, and n and f are the respective distances to the near and far plane along the z-axis. The field of view can, in turn, be expressed as a function of the focal length f' and the size of the sensor h, both in millimetres:

$$\theta_v = 2 \tan^{-1}\left(\frac{h}{2f'}\right). \tag{3}$$

This relation can be utilised to configure the virtual camera to match a physical one with known, or estimated, focal length and sensor size.

Vertices that project outside the canonical view volume are discarded in a process called *clipping*. This eliminates erroneous projections from geometry located behind the camera and reduces the amount of geometry to be processed in subsequent steps. Furthermore, the z coordinates are kept in a *depth buffer*, which is used to keep track of the relative distance to the virtual camera of every vertex. This is later used to resolve conflicts where geometry is projected to the same location on the screen.

The penultimate step is to transform the canonical space into screen space, using the *viewport transformation matrix* [21]:

$$M_D = \begin{bmatrix} \frac{W}{2} & 0 & 0 & X + \frac{W}{2} \\ 0 & \frac{H}{2} & 0 & Y + \frac{H}{2} \\ 0 & 0 & \frac{F-N}{2} & \frac{F+N}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{4}$$

where W is the width of the viewport, H is its height and (X, Y) are the coordinates of its bottom-left corner, all in pixels. N and F define the new near and far plane, and are typically 0 and 1, respectively.

Finally, the projected triangles are rasterised into *fragments*, which are points in screen space that may contribute to the final pixel values. Traditionally, a fragment belongs to a triangle if its centre lies within it, as illustrated in Fig. 3. This is the stage where colour information from potential textures is computed, resulting in a colour of each fragment. This is performed by *fragment shaders*, which are programs that run on the graphics processor for each fragment. Multiple triangles can overlap when projected into the virtual camera, leading to their fragments occupying the same parts of the screen. In such cases, the depth buffer determines which fragment controls the final colour of the pixel. Typically, the fragment closest to the camera is chosen. However, in the case of translucency, multiple fragments may contribute to the same pixel.

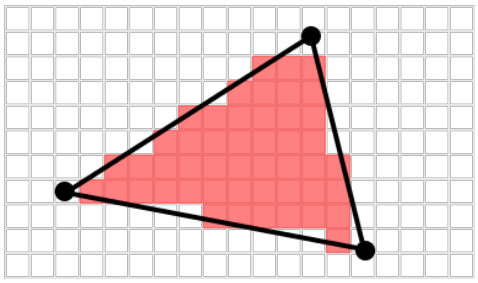
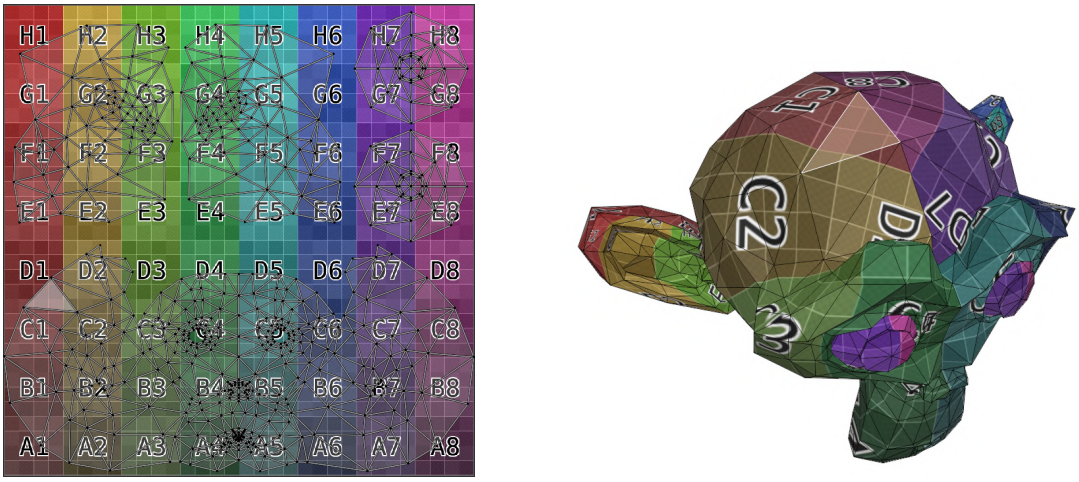


Fig. 3. Projected triangle rasterised into coloured fragments.

2.1.2 Texture Mapping

To assign colour information to a mesh, one may utilise *albedo* textures, also referred to as colour textures. These are regular colour images, often with square aspect ratios. Every pixel, or *texel*, of the texture contains a colour to be sampled when rendering the corresponding part of the mesh. The translation between texels and the mesh is handled by *texture mapping* (also referred to as UV mapping). This requires the assignment of two-dimensional UV coordinates, which are normalised between 0 and 1 in texture space, to every vertex of the mesh. When a triangle is rasterised by the computer's graphics processor, the colours of its fragments are sampled from the albedo texture by interpolating the UV coordinates of the three vertices making up the face (see Fig. 4).



(a) Vertices mapped to two-dimensional UV coordinates in texture space.

(b) Texture mapped to a mesh by interpolating the vertices' UV coordinates.

Fig. 4. Texture mapping. The same triangle is highlighted in both spaces.

This interpolation is typically performed using barycentric coordinates, in a two-step process. First, the barycentric coordinates (w_1, w_2, w_3) of the point $\mathbf{p}$ on the triangle $\mathbf{f} = (\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3)$ are determined such that $\mathbf{p} = \sum_{i=1}^{3} w_i \mathbf{v}_i$. An extra normalisation constraint $\sum_{i=1}^{3} w_i = 1$ ensures that the solution is unique and that the coordinates can be used as proportions of attributes of the three vertices. Then, these coordinates are used to compute the point's UV coordinates (u, v) as a linear combination of the three corner vertices' UV coordinates (u_i, v_i) using:

$$u = w_1 u_1 + w_2 u_2 + w_3 u_3, \tag{5}$$

and

$$v = w_1 v_1 + w_2 v_2 + w_3 v_3. \tag{6}$$

In addition to colour information, graphics pipelines often support other types of textures, or *maps*, serving different purposes. A few common ones are *normal*, *bump*, *metallic*, *roughness*, and *ambient occlusion* maps. Instead of storing colour information, their texels define attributes such as the fine structure of the mesh's surface, or material characteristics such as how it tints reflected light.

2.1.3 Differentiable Rendering

Differentiable rendering refers to rendering pipelines whose output is differentiable with respect to the scene parameters. This enables the use of gradient-based methods to optimise those parameters — such as a mesh's vertex positions, the texels of its textures, or the properties of the virtual camera — to minimise a loss function expressed on the rendered image.

While certain operations in traditional rendering pipelines are inherently differentiable, such as the transformations and projections outlined in Section 2.1.1, others require modification [22]. For example, the derivative of the colour of a fragment belonging to a uniformly coloured triangle is always zero with respect to that triangle's vertex positions; tiny perturbations to a vertex's position will not have any effect on any of the triangle's fragments. However, the translation of vertices may reveal previously occluded geometry, resulting in a discontinuous step change in the pixel colour. In response, differentiable rendering pipelines utilise techniques to reformulate either the rendering process or its differentiation.

One such approach is the local approximation of gradients by applying filters to the rendered image, allowing nearby pixels to influence each other. Loper and Black [23] identify pixels at the boundaries between two surfaces using the depth buffer, and approximate the gradient at those points using vertical and horizontal filters of the form $\frac{1}{2}[-1, 0, 1]^\top$ and $\frac{1}{2}[-1, 0, 1]$, respectively. This enables non-zero gradients at boundaries, indicating a smooth change in pixel colour as a function of lateral translation of the occluding surface. Naturally, this formulation does not support gradients for triangles' z coordinates in camera space.

Another technique is to render the image using a relaxed depth buffer, such that an aggregate of all overlapping fragments controls the final pixel colour instead of only the one closest to the camera. Such an aggregation eliminates discontinuities caused by depth-wise translation. Similarly, to address the problem of discontinuities from lateral translation of faces, the hard face boundaries can be smoothed out to become continuous. For example, Liu *et al.* [24] combine these two ideas by using a probability map $\mathcal{D}_j$ for the influence of a projected triangle $\mathbf{f}_j$ over any pixel $\mathbf{p}_i$:

$$\mathcal{D}_j^i = \text{sigmoid}\left(\delta_j^i \cdot \frac{d^2(i,j)}{\sigma}\right), \tag{7}$$

where δ_j^i is $+1$ if $\mathbf{p}_i$ lies inside $\mathbf{f}_j$, but -1 otherwise; $d(i,j)$ is the shortest distance between pixel $\mathbf{p}_i$ and any edge of $\mathbf{f}_j$; and σ is a small positive scalar controlling the sharpness of the smoothed edge. The result of such a modification is visualised in Fig. 5.

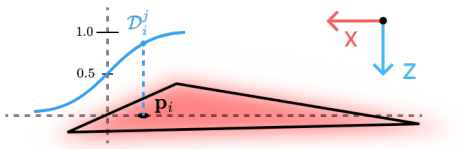


Fig. 5. Side view illustrating how the boundaries of the projected triangle are smoothed by the sigmoid function to avoid discontinuities between surfaces.

The final pixel colour I^i is then computed as a weighted sum over the overlapping fragments combined with a background colour:

$$I^i = \mathcal{A}_S(\{C_j\}) = \sum_j w_j^i C_j^i + w_b^i C_b, \tag{8}$$

where C_j^i is the colour of triangle j at pixel i and C_b is the background colour. The weights w_j^i and w_b^i are computed using a softmax with temperature $\gamma > 0$, where the terms are scaled by the respective triangle's influence $\mathcal{D}_j^i$:

$$w_j^i = \frac{\mathcal{D}_j^i \exp(z_j^i/\gamma)}{\sum_k \mathcal{D}_k^i \exp(z_k^i/\gamma) + \exp(\epsilon/\gamma)}. \tag{9}$$

The temperature γ controls the sharpness of the distribution of weights, and the use of $\epsilon > 0$ ensures that $w_b^i \equiv 1 - \sum_j w_j^i > 0$ to always assign some positive weight to the background colour.

This thesis uses PyTorch3D [25], which is a modular differentiable renderer built on top of the PyTorch [26] deep learning framework. It supports mesh and point cloud rendering, and takes inspiration from [24] to overcome the issues mentioned above. Specifically, it aggregates the closest K faces in the z direction (as opposed to including all faces) for depth-wise gradients, and uses a blur radius for vertical and horizontal gradients. In contrast to [24], this radius sets an upper bound on the range of influence a triangle can have over pixels.

2.2 Tools for 3D Development

To construct the 3D environment in which the match will be recreated, two primary 3D tools are required. First, Blender [27] is used to create all 3D graphics, followed by the game engine Unity [28], which will be utilised to develop the interactive experience that will run on the VR headset.

2.2.1 Blender

Blender is an open-source 3D creation suite that supports the entirety of the 3D pipeline, including modelling, rigging, animation, simulation, rendering, compositing, and motion tracking.

One of Blender's key strengths is its advanced rigging and animation capabilities, essential for creating lifelike character movements. Tools such as the Armature object enable complex skeletal rigging, allowing for precise control over character joints and movements. Blender also utilises Quaternions for 3D rotations, which help in avoiding the common problem of gimbal lock, thereby providing smooth and continuous rotations.

2.2.2 Unity

Unity is a cross-platform game engine that has become synonymous with interactive content development. Unity has been widely adopted for the development of video games, simulations, and other real-time 3D applications.

A significant development in Unity is its support for creating Virtual Reality (VR) environments. Unity provides comprehensive tools for VR development, allowing developers to create immersive and interactive experiences. This includes support for various VR headsets and motion controllers, enabling realistic and engaging user interactions within virtual spaces.

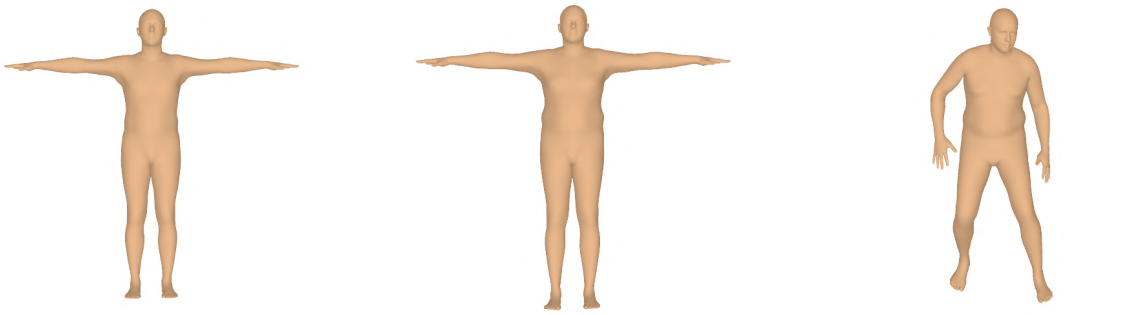
In addition to VR, Unity is highly effective in creating interactive 3D environments for specific applications, such as sports simulations. As in this thesis, Unity can be used to develop interactive 3D environments for football, where users can experience realistic gameplay scenarios. The engine's physics and rendering capabilities allow for the creation of detailed stadiums, player models, and dynamic interactions.

2.3 Skinned Multi-Person Linear (SMPL) Model

The Skinned Multi-Person Linear (SMPL) model [29] is a learned, vertex-based model of human body shapes and poses. It consists of $N = 6890$ vertices and $K = 23$ joints. There is a template pose $\bar{\mathbf{T}} \in R^{3N}$ defining the position of all vertices in a rest pose. This is transformed into

$$T_P(\vec{\beta}, \vec{\theta}) = \bar{\mathbf{T}} + B_S(\vec{\beta}) + B_P(\vec{\theta}) \tag{10}$$

where $B_S(\vec{\beta})$ and $B_P(\vec{\theta})$ are two sets of blend shapes, which shift the template pose vertices based on the identity-dependent shape parameters $\vec{\beta} \in \mathbb{R}^{10}$, and the pose $\vec{\theta} \in \mathbb{R}^{72}$, respectively. The deformed template pose vertices are then articulated using linear blend skinning (LBS) according to the pose $\vec{\theta}$. See Fig. 6 for a visualisation of these steps.



Template pose ($\bar{\mathbf{T}}$) $\bar{\mathbf{T}} + B_S(\vec{\beta})$ $W(T_P(\vec{\beta}, \vec{\theta}), \mathbf{J}(\vec{\beta}), \vec{\theta}, \mathcal{W})$

Fig. 6. Deformation of SMPL mesh in multiple stages.

LBS is a method for articulating the vertices of a mesh based on a given pose. In the general case, every vertex $\bar{\mathbf{t}}_i$ of the rest pose is transformed as a function of the pose $\vec{\theta}$, the joint locations $\mathbf{J}$, and a set $\mathcal{W}$ of skinning weights $w_{k,i}$ controlling the influence of joint k over vertex i:

$$\bar{\mathbf{t}}'_i = \sum_{k=1}^{K} w_{k,i}\, G'_k(\vec{\theta}, \mathbf{J})\, \bar{\mathbf{t}}_i. \tag{11}$$

The vertex translations are thus weighted sums of all joint transformations $G'_k(\vec{\theta}, \mathbf{J})$, scaled by the joint-to-vertex skinning weight $w_{k,i}$. In the case of SMPL, the equation is extended to first deform the rest-pose vertices based on the identity-dependent shape parameters $\vec{\beta}$ and the pose-dependent blend shapes. Additionally, the joint positions $\mathbf{J}$ are estimated as a function of the body shape parameters $\vec{\beta}$. This results in the following modified skinning equation:

$$\bar{\mathbf{t}}'_i = \sum_{k=1}^{K} w_{k,i}\, G'_k(\vec{\theta}, \mathbf{J}(\vec{\beta}))\, (\bar{\mathbf{t}}_i + \mathbf{b}_{S,i}(\vec{\beta}) + \mathbf{b}_{P,i}(\vec{\theta})). \tag{12}$$

2.4 Diffusion Models (DM)

One of the most prolific and versatile methods for text-to-image and image-to-image generation is the utilisation of denoising diffusion models [15]. Conceptually, these are trained to identify and remove varying amounts of noise added to images from the training data. Once trained, they can produce arbitrary images by successively removing the predicted noise of the input, starting from a sample of pure Gaussian noise.

There is a forward process and a learned reverse process, as illustrated in Fig. 7. The forward process gradually corrupts an image $\mathbf{x}_0$ from the training set with Gaussian noise $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ to form intermediate latent representations $\mathbf{x}_t$ of the same dimension as $\mathbf{x}_0$:

$$\mathbf{x}_t = \sqrt{1 - \beta_t}\mathbf{x}_{t-1} + \beta_t \boldsymbol{\epsilon}. \tag{13}$$

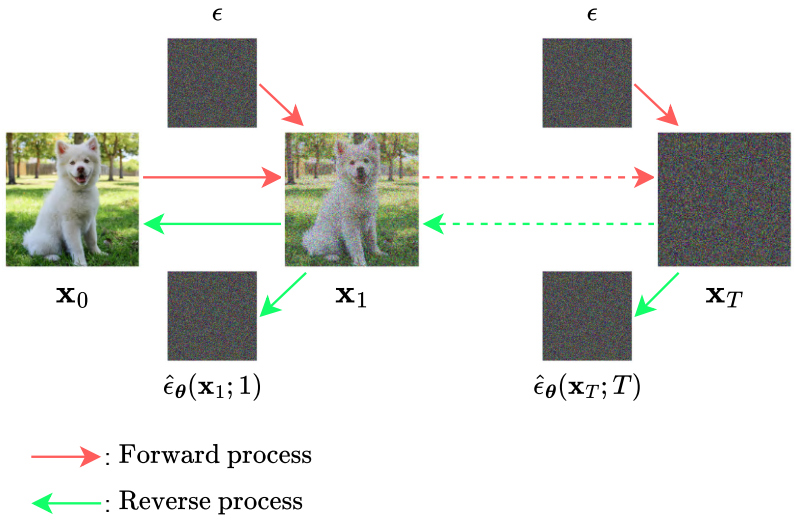


Fig. 7. Forward process that gradually corrupts a sample with Gaussian noise, and reverse process that removes the predicted noise.

Governing this process is a noise schedule, or variance schedule, consisting of a sequence of fixed scalars $\beta_1, \ldots, \beta_T$ which define the number of time steps T as well as the balance between signal and noise at each step.

The reverse process utilises a learned denoiser $\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\cdot)$ to generate predictions of the noise contents of the corrupted input $\mathbf{x}_t$. Subtracting this from the input and repeating the process in an autoregressive manner gradually increases the likelihood of the sample under the estimated distribution of the training data.

Each iteration of the training algorithm starts by sampling an image $\mathbf{x}_0$ from the training data, a time step $t \sim \mathcal{U}(\{1, \ldots, T\})$, and Gaussian noise $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. The forward process corrupts the image from $\mathbf{x}_0$ to $\mathbf{x}_t$ in a single step using the closed-form formula:

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\epsilon}, \tag{14}$$

where $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$ and $\alpha_t \equiv 1 - \beta_t$. The denoiser then computes $\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\mathbf{x}_t; t)$ to estimate the added noise. Finally, its parameters $\boldsymbol{\theta}$ are updated by following the negative gradient of the mean squared error loss between the actual and the predicted noise:

$$\|\boldsymbol{\epsilon} - \hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\mathbf{x}_t; t)\|^2. \tag{15}$$

Denoising diffusion models are sequential in nature; inference from random noise to a final image takes many steps, each of which requires evaluating the denoiser to generate a prediction, making the process very resource-intensive.

2.4.1 Latent Diffusion Models (LDM)

A variant of the diffusion model is the *latent diffusion model* (LDM) [30], which aims to address the computational challenges with regular diffusion models. Instead of perturbing the pixels of the image directly, it is first embedded into a latent space using a pre-trained image encoder $\mathcal{E}$:

$$\mathbf{z}_0 = \mathcal{E}(\mathbf{x}_0). \tag{16}$$

The training process is identical apart from being performed on image latents as opposed to pixels. Since this latent space is lower-dimensional than the image space, LDMs tend to be faster to run and less memory-intensive to train. Finally, at the end of the reverse process, the denoised latents are converted back to image space using a pretrained image decoder $\mathcal{D}$:

$$\hat{\mathbf{x}}_0 = \mathcal{D}(\hat{\mathbf{z}}_0). \tag{17}$$

2.4.2 Text-Conditioned Diffusion Models

A common application of diffusion models is that of text-conditioned image generation. This conditioning is typically achieved in one of two ways. One is *classifier guidance* [31], which trains a separate image classifier $p_{\boldsymbol{\phi}}(\cdot)$ in tandem with an unconditional diffusion model. During inference, the predicted noise from the denoiser $\hat{\boldsymbol{\epsilon}}_\theta(\mathbf{x}_t, t)$ is augmented with gradients $\nabla_{\mathbf{x}_t} \log p_{\boldsymbol{\phi}}(\mathbf{y}|\mathbf{x}_t)$ from the classifier to maximise the likelihood of the provided class $\mathbf{y}$. The downside of this setup is that it requires an additional model. Furthermore, this cannot simply be a pre-trained off-the-shelf model since it has to be trained on noisy samples.

In contrast, the more widely used *classifier-free guidance* (CFG) [32] directly conditions the diffusion model itself on embeddings $\mathbf{y}$ generated from a text prompt. In the case of Stable Diffusion [33], an LDM, these embeddings are generated with a transformer and are integrated into the denoiser using cross-attention [34]. With CFG, the predicted noise is computed as a linear combination of the conditional $\hat{\boldsymbol{\epsilon}}_\theta(\mathbf{x}_t; \mathbf{y}, t)$ and unconditional $\hat{\boldsymbol{\epsilon}}_\theta(\mathbf{x}_t; \emptyset, t)$ output controlled by the *guidance scale* ω:

$$\hat{\boldsymbol{\epsilon}}_\theta^{CFG}(\mathbf{x}_t; \mathbf{y}, t) = (1+\omega)\hat{\boldsymbol{\epsilon}}_\theta(\mathbf{x}_t; \mathbf{y}, t) - \omega\hat{\boldsymbol{\epsilon}}_\theta(\mathbf{x}_t; \emptyset, t). \tag{18}$$

The guidance scale constitutes a tunable trade-off between the quality (large ω) and the diversity (small ω) of the outputs.

2.4.3 Image-Conditioned Diffusion Models

Another application of diffusion models is that of image-to-image generation. This involves conditioning the model on some starting image, possibly in addition to a text prompt. Concretely, the image conditioning can be achieved by simply concatenating the starting image with the noisy sample before feeding them into the model. Image-to-image generation has been used for image super-resolution, colourisation of black-and-white images, infilling of occluded parts of images (inpainting), and generation of surrounding pixels given a cropped-in image (outpainting).

Inpainting is the task of filling in masked-out regions of an image with plausible content. One approach popularised by Saharia *et al.* [35] is to replace the masked regions with Gaussian noise while retaining the original, uncorrupted sample for the remaining parts of the image. The model is then trained by computing the loss only for the regions to be inpainted. This technique maintains a general model architecture that also supports other image-to-image tasks.

2.5 Diffusion Models for Texture Reconstruction

Previous work on human texture reconstruction has utilised denoising diffusion models in various forms. The approaches include both image inpainting and image-to-image enhancements, which have been applied both in texture space and to the image. This section highlights previous works that either directly relate to human texture generation, or which can be modified to do so.

2.5.1 Image Inpainting

A recent example is the work by Ho *et al.* [16], which approached the problem of single-view textured human reconstruction by training a latent diffusion model to hallucinate back-view images of a person given only a captured front-view image of them. To ensure the back-view image is both visually and spatially consistent with the front-view one, they condition the LDM on CLIP [36] latents extracted from the front-view image. They also augment its weights with a ControlNet [37] model conditioned on two additional inputs. One is a mask of the person, estimated from the front-view image. It is flipped horizontally to become the back-view counterpart. This mask is generated by the pre-trained Segment Anything Model (SAM) from Kirillov *et al.* [38]. The other input to the ControlNet is the UV coordinates of the person's back side, which are taken from a back-view rendering of an SMPL mesh in the estimated pose of the person. The pose is estimated using SMPLer-X [39]. The result is an algorithm that estimates both the geometry and the clothing of a person from a single front-view image.

Another related work is Human-SGD from AlBahar *et al.* [17], which also tackles the problem of human mesh and texture reconstruction from a single view. Their approach is instead to aggregate colour information synthesised from multiple views. They initially hallucinate a back-view image, and then successively rotate the posed estimated mesh and inpaint newly exposed parts using a pre-trained diffusion model. In addition to the rendered image and the mask of the regions to inpaint, this model is conditioned on a silhouette mask of the entire person and the surface normals of the estimated mesh. This guides the inpainting process to adhere both to the overall pose and to the structure of the surface.

The work of Casas and Comino-Trinidad [40] introduces SMPLitex, a human texture generation algorithm centred around a Stable Diffusion model fine-tuned on 250 SMPL-formatted UV textures. The goal is to estimate a full colour texture of a person from a single view. It uses DensePose from Güler *et al.* [41] for predicting pixel-to-surface correspondences, assigning a UV coordinate to every pixel in the image that belongs to the human subject. Additionally, they use Semantic Guided Human Matting [42] to mask out the person in the image.

Combining the two off-the-shelf models, the first step of the algorithm produces a partial texture of the human by directly mapping the pixels of the image to the texels at their predicted UV coordinates. Next, the remaining pixels are filled in using the SMPLitex diffusion model, conditioned on the partial texture, and a generic text prompt, to hallucinate plausible detail given the parts that were visible in the image. Finally, as a last step, they perform an image-to-image pass to improve the quality of the whole image, using the same model.

2.5.2 Score Distillation Sampling (SDS)

To circumvent the lack of available labelled 3D data, Poole *et al.* [43] invented a novel approach to text-to-3D generation termed Score Distillation Sampling (SDS). It entails training a neural radiance field (NeRF) [7] model to synthesise 3D scenes by distilling the knowledge of a pre-trained 2D image diffusion model, specifically Imagen from Saharia *et al.* [44]. In every iteration of the training loop, the NeRF model renders the scene from a random direction, producing $\mathbf{x} = g(\boldsymbol{\phi})$, where $\boldsymbol{\phi}$ are the parameters of the NeRF model which are to be optimised. This rendered image is then corrupted with Gaussian noise $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ to form $\mathbf{x}_t$, and the pretrained denoising diffusion model then predicts the noise contents of this corrupted image $\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}^{CFG}(\mathbf{x}_t; \mathbf{y}, t)$, conditioned on embeddings from a text prompt $\mathbf{y}$. The actual noise is subtracted from the predicted noise to reduce variance, forming a *noise residual* which is used as the gradient to be propagated back to the NeRF model's parameters $\boldsymbol{\phi}$ using the chain rule. Mathematically, the full gradient is defined as:

$$\nabla_{\boldsymbol{\phi}} \mathcal{L}_{SDS}(\boldsymbol{\theta}, \mathbf{x} = g(\boldsymbol{\phi})) \equiv \mathbb{E}_{t,\boldsymbol{\epsilon}} \left[w(t) (\hat{\boldsymbol{\epsilon}}_{\theta}^{CFG}(\mathbf{x}_t; \mathbf{y}, t) - \boldsymbol{\epsilon}) \frac{\partial \mathbf{x}}{\partial \boldsymbol{\phi}} \right], \tag{19}$$

where $w(t)$ is a scaling term that incorporates the constant gradient of $\mathbf{x}_t$ with respect to the rendered image $\mathbf{x}$, and $\frac{\partial \mathbf{x}}{\partial \boldsymbol{\phi}}$ are the gradients of the rendered image with respect to the NeRF model's parameters.

Notably, this formulation omits the gradients of the predicted noise $\hat{\boldsymbol{\epsilon}}_{\theta}^{CFG}(\mathbf{x}_t; \mathbf{y}, t)$ with respect to the noisy sample $\mathbf{x}_t$. This gradient is expensive to compute since it requires backpropagation through the denoiser, and the authors observe that leaving it out improves the realism of the generated samples.

One can reformulate this as a loss function $\mathcal{L}_{SDS}$ consisting of the inner product between the noise residual and the rendered image, scaled by $w(t)$. An auto-differentiation framework such as PyTorch would generate the same gradient as in Eq. (19) during the backward pass of this loss. An illustration of this training setup is shown in Fig. 8.

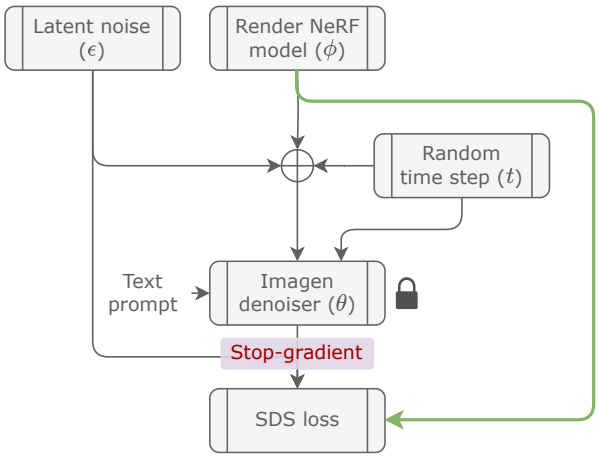


Fig. 8. Score Distillation Sampling where a NeRF model is trained by distilling knowledge from a pretrained 2D diffusion model. The green path shows how gradients flow from the loss to the NeRF model. No gradients flow through the denoiser.

3

System Overview

In this chapter, an overview of the entire system is presented. The aim is to provide a clear and simplified picture of all processes, making it easier to understand more detailed information later on. Firstly, an overview of the system's pipeline is presented, including the other two master's theses. Then, a more detailed pipeline for building the VR environment is provided, followed by a description of the model used for texture generation.

3.1 Visual Representation of the Entire System

As described, the entire system is partly based on two other works. The first work continuously identifies all players and the ball in each frame. The second work estimates the physical pose and location of each player in 3D space. This master's thesis (the third work) focuses on creating a visual and interactive representation of the system. Together, this can be visualised as a pipeline in Fig. 9.

(a) Original video from match between IFK Göteborg – AIK.

(b) The first work tracking all players on the field including the ball.

(c) The second work estimating the pose for each player with help from the tracked players from the first work.

(d) Taking the tracked and posed players and ball into a 3D environment and reconstructing the match.

Fig. 9. Visual representation of the pipeline from video to the last step where the football match is reconstructed in 3D.

The pipeline in Fig. 9 shows a straightforward approach, where one may think that everything happens sequentially. However, this is not the whole picture. The complete flow of information is visualised in Fig. 10. Everything starts with the video clip, which is broken down into individual frames. These frames are then fed to the player- and ball-tracking model (for which the first work is responsible). The first frame of the clip is also fed to the camera calibration model (the second work's responsibility) which outputs the camera calibration parameters.

These steps happen in parallel. When they are finished, the output from the player tracking model in the form of bounding boxes is fed to the pose estimation model. Then, the produced pose estimates are converted to a format that is compatible with Unity. The pose estimates in combination with the camera parameters are given to the texture estimation model, which estimates the players' textures. Lastly, all this information — the video, camera parameters,

Unity-converted pose estimates, ball tracking, and player textures — are given to Unity which creates the visual and interactive VR experience.

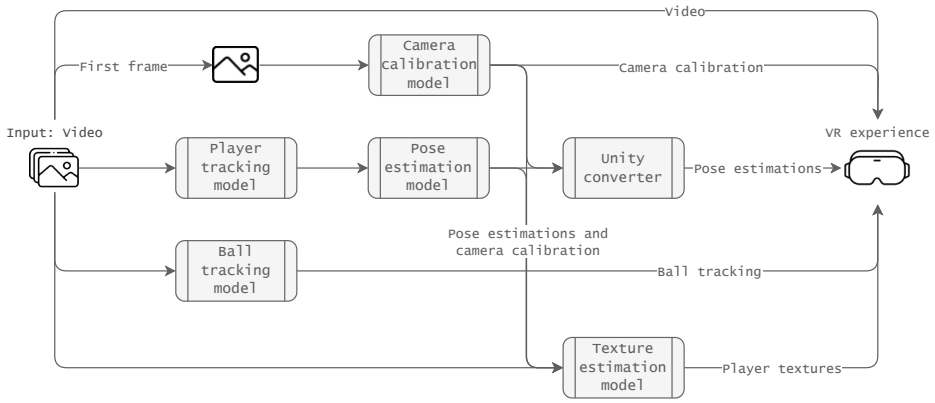


Fig. 10. Visualisation of entire system architecture.

3.2 VR Environment Architecture

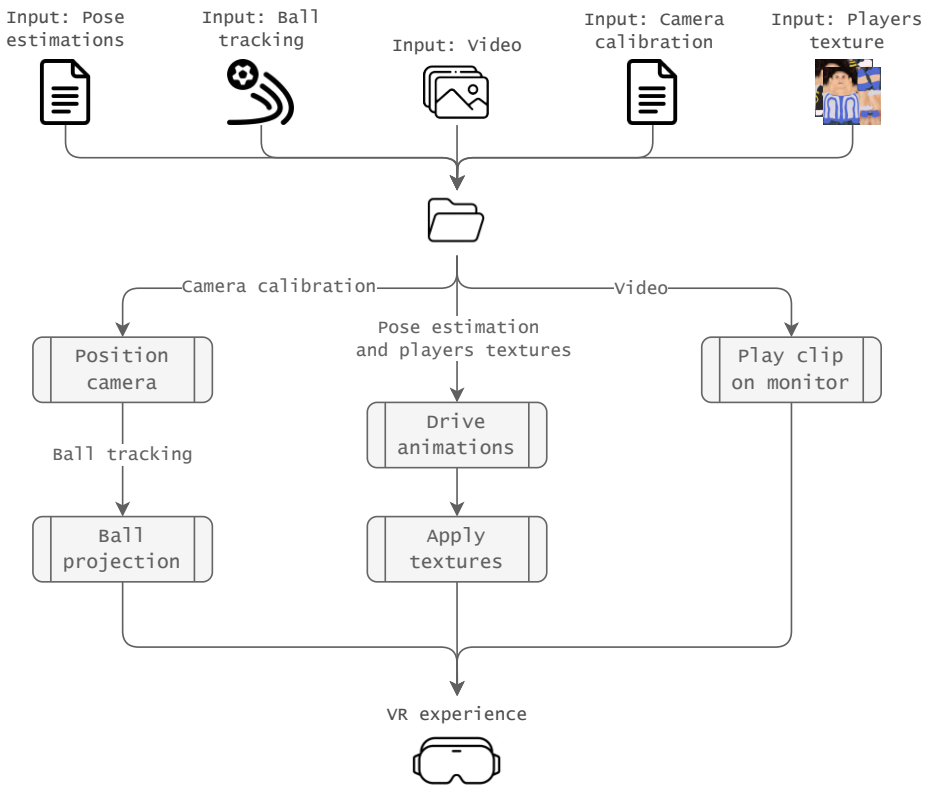


Fig. 11. Visualisation of VR environment architecture.

As shown in Fig. 10, the VR environment sits at the end of the pipeline architecture. It uses more or less all previous inputs and outputs as the other works have produced and combines them into an interactive VR experience. Fig. 11 is a visualisation of the VR environment architecture.

First, the input files — pose estimates (from the second work), ball tracking (from the first work), the video clip, the camera parameters (from the second work) and the players' textures (from this master's thesis) — are put into a combined folder.

Then, inside Unity, three things happen. First, the camera is positioned, rotated and configured according to the estimated camera parameters. The ball is then projected down to the pitch from the view of the camera. Second, the pose estimates are used to drive the players' animations, and then the textures are applied to the players. Lastly, the video clip is attached to a screen in the VR environment to illustrate the original scene.

3.3 Differentiable Renderer Architecture

The end goal of the differentiable rendering pipeline is to create individual high-quality UV texture maps which can be applied to each player. The differentiable renderer is dependent on the first and second work. From the first work, we are given tracking data of the players in the form of bounding boxes. From the second work, we get the articulated 3D pose estimates in the form of SMPL pose and shape parameters, and the camera calibration for that clip. The pipeline is shown in Fig. 12.

The initial input is a video, which then is decomposed into individual frames. Utilising the tracking data from the first work, these frames are cropped into patches centred on individual players. These patches are used to compare with the pose estimates, by aligning the 3D models accurately to the corresponding players. The patches can also be referred to as the targets of the model.

Then, with the help of the SMPL-X Python package [45], the SMPL parameters are converted into 3D meshes that are compatible with PyTorch3D. The SMPL models are all rendered (initially with a uniform grey texture) in a virtual camera configured according to the camera calibration obtained from the second work.

Finally, the rendered meshes are compared with the targets patches, producing a loss which is used to update the texture. However, only the pixels corresponding to the clothes of the rendered player are updated (using a *posed clothing mask*). Then the updated pixels are combined with the starting texture and texture mask to ensure no pixels are updated except the clothes.

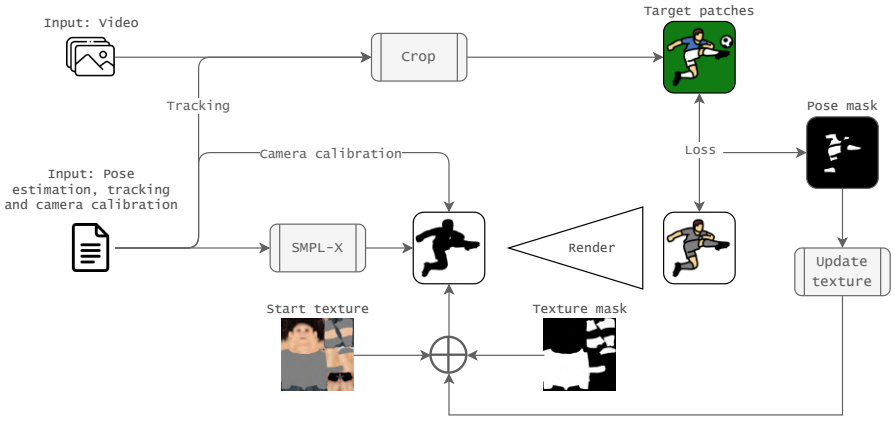


Fig. 12. Visualisation of differentiable rendering architecture.

4

Creating a VR Environment for Football Reconstruction

In this chapter, the process of creating the 3D environment to replicate the recorded match is described. More specifically, it details the construction of the 3D environment, the simulation of the ball, and the steps taken to make it interactive. Finally, some results from the environment are presented.

4.1 Input Files for 3D Reconstruction

The simulation is driven by data from a video clip of a football match. The data consists of four input files, as outlined in Table I. All files belonging to a given video clip are placed in a folder on the VR headset's file system, as illustrated in Fig. 11. They are parsed at runtime, thus enabling switching between different clips without recompiling the game.

TABLE I. Description of the data files that Unity receives and their data format.

Input Data	**Data Format**
Estimated pose sequences of all players	`.glb`
Extracted image textures for all players	`.png`
Estimated camera calibration parameters	`.json`
Sequence of 2D pixel coordinates of the tracked ball	`.csv`

The central piece of this data is a GLTF-formatted file (`.glb`) containing all tracked players' pose sequences throughout the video clip. Specifically, for each player, there is an animation clip specifying their body pose and location on the pitch for every frame they were in view of the camera. The animation data is applied to an SMPL mesh for a visual representation of the player. Additionally, every player's mesh is assigned a unique colour texture (`.png`) representing their kit. The generation of those textures is described in greater detail in Chapter 5.

Finally, there is a text file (`.csv`) specifying the location of the ball in camera space for every frame of the video clip. Every line contains the two-dimensional pixel coordinates of the centre of the ball for the corresponding frame. This is paired with a file (`.json`) containing

the estimated parameters of the camera that captured the video clip. When combined, they enable the projection of the ball onto the pitch, as described in Section 4.2.1.

4.2 Building the 3D Football Environment

The virtual environment consists of a 3D-modelled full-size football pitch enclosed in a spherical 360-degree image of a football stadium. There is a floating spectator booth serving as a starting point for the user, from which they can view and interact with the reconstructed match (see Fig. 13). One such interaction is the teleportation onto the pitch to experience the match from the view of one of the players.

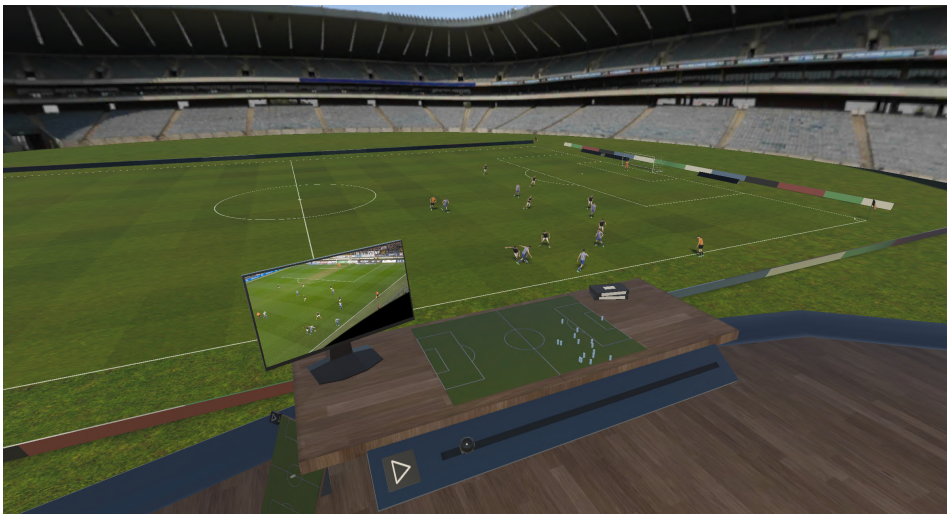


Fig. 13. VR environment as viewed from the floating spectator booth.

The pitch itself is modelled in Blender. It consists of a rectangle of 105 metres by 68 metres, painted with a repeating grass texture and tinted with horizontal and vertical stripes. The lines and markings are modelled as a white, flat 3D mesh, making them easy to adapt to pitches of any size.

The spectator booth is modelled in the vein of a flying saucer with a panorama view over the whole pitch. There is a table with a monitor showing the original clip and a tactical view of the pitch with markers for the players and ball. Showing the original clip informs the user about the accuracy of the reconstruction. The table also contains a timeline for scrubbing through the clip, a play/pause button, as well as interactive cassette tapes for selecting the clip to be analysed.

4.2.1 Simulating the Ball

The ball is tracked in the image plane to produce a sequence of 2D-pixel coordinates. To render it on the pitch, these 2D coordinates must be converted into 3D coordinates in world space. This is achieved by projection from the camera's point of view, utilising the estimated camera intrinsics and extrinsics. Specifically, a ray is cast from the camera's location $\vec{t}$ in world space, in the direction $\vec{b}$ of the ball extending until it hits the plane parallel to the

ground plane, offset upwards by the ball's radius r where the centre of the 3D model of the ball then is placed.

First, each 2D coordinate is normalised by dividing by the resolution ($W \times H$ pixels) of the video clip. It is then recentred such that both components lie in the interval $[-0.5, 0.5]$:

$$x_{\text{norm}} = \frac{x}{W} - 0.5, \tag{20}$$

$$y_{\text{norm}} = \frac{y}{H} - 0.5. \tag{21}$$

Then, the azimuth and elevation offsets, $\Delta\theta_{\text{azim}}$ and $\Delta\theta_{\text{elev}}$, of the ray toward the ball from the camera's forward vector, can be computed as fractions of the estimated horizontal (θ_h) and vertical (θ_v) field of view of the camera, respectively:

$$\Delta\theta_{\text{azim}} = x_{\text{norm}} \cdot \theta_h, \tag{22}$$

$$\Delta\theta_{\text{elev}} = y_{\text{norm}} \cdot \theta_v. \tag{23}$$

The camera's forward direction is panned by $\Delta\theta_{\text{azim}}$, then tilted by $\Delta\theta_{\text{elev}}$ to point toward the ball, forming the ball ray direction $\vec{b}$ of unit length.

Finally, to compute the extent of this ray to reach the pitch, we solve for λ in the below equation,

$$r = \vec{t}_y + \lambda \vec{b}_y, \tag{24}$$

producing the expression for the position of the ball:

$$\vec{t} + \left(\frac{r - \vec{t}_y}{\vec{b}_y} \right) \vec{b}. \tag{25}$$

This process is visualised in Fig. 14.

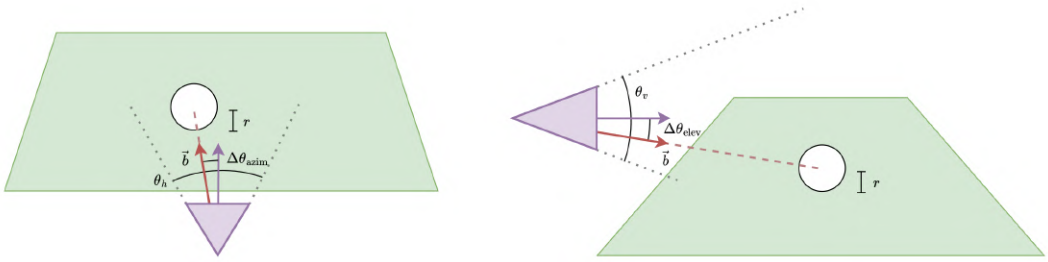


Fig. 14. Visualisation of ball projection from two angles.

A natural limitation of this approach is the depth ambiguity incurred by the use of a single camera. The ball is assumed to be at or near the ground at all times; should it travel high above

the ground, the camera ray will travel farther and the ball would be assigned an erroneous position that is offset away from the camera.

The ball is continuously rotated about its axis to roll in its current direction of travel. The rotation $\Delta\phi$ (in radians) during a given frame is a function of its displacement $\Delta\vec{x}$ from the previous frame:

$$\Delta\phi = \frac{\|\Delta\vec{x}\|_2}{r}, \tag{26}$$

where $\|\Delta\vec{x}\|_2$ is the ℓ_2 norm of the displacement vector in meters, and r is the radius of the ball, also in meters.

4.2.2 Players' Football Kits

The players' football kits are represented as albedo textures that are UV mapped to their SMPL 3D models. The generation of those textures is detailed in Chapter 5. The assignment of textures to players is automated, utilising the fact that pose sequences are read from the GLTF file in order of ascending player ID and that each texture is tagged with the ID of the player it refers to.

4.3 Adding Interactivity with the Meta Quest 3

This project aimed to create an environment which the user can step into and experience from the point-of-view of the players. A virtual reality headset is a perfect fit for making this experience immersive and intuitive. The Meta Quest 3 [46] makes it accessible by performing all tracking and compute onboard the headset, eliminating the need for external tracking cameras or tethering to a PC that runs the game.

4.3.1 Creating Interactive Elements

The user can interact with the reconstructed game by controlling playback, and by selecting the viewpoint from which to observe the game, whether as a bystander or through the perspective of a player. Furthermore, they should have this control both in the spectator booth and on the pitch. To enable this, two variants of a control panel are developed: one is in the form of a table in the spectator booth (shown in Fig. 13), and the other is a handheld tactics board that is anchored to the user's left arm when out on the pitch. They both have the same set of controls, namely a timeline with a handle for scrubbing, a play/pause button, and a map of the pitch with markers for the players and the ball. With this map, the user may teleport to any place on the pitch, or specify a player from whose perspective they want to view the game. The tactics board is shown in Fig. 15.

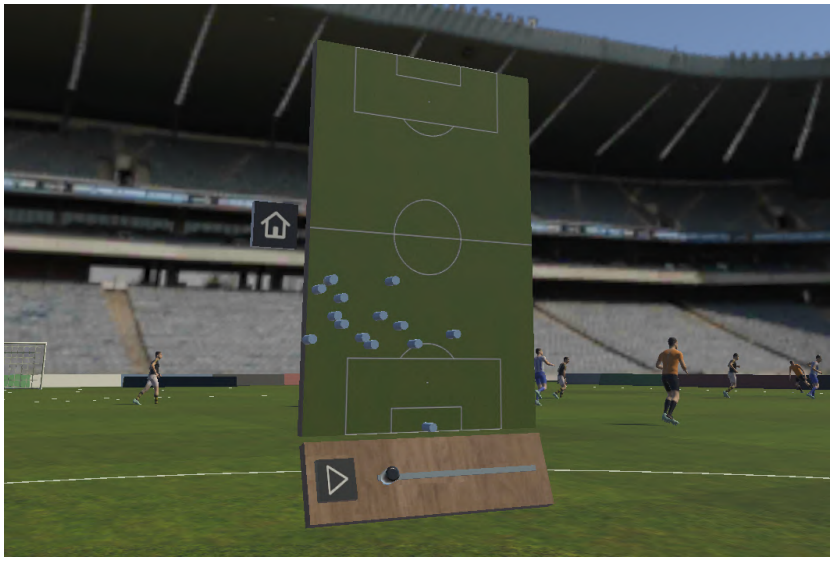


Fig. 15. Handheld tactics board with controls for time and position. It consists of a map with markers for the players and the ball, a timeline for scrubbing through time, and a play/pause button.

4.3.2 VR Performance Optimisations

Running the game completely onboard the Meta Quest 3 headset poses resource constraints inherent to mobile platforms. If not handled correctly, these may result in below-acceptable frame rates and incur a latency between user inputs and visual feedback. In turn, this may induce motion sickness. To mitigate these, strides were taken to ensure the game was running as smoothly as possible.

Most of the environment remains static throughout the game's runtime, except the players, the ball, and a few interactive elements. Consequently, it is suitable to use textures with precomputed lighting information instead of real-time lighting. This procedure, known as light baking, involves performing a one-time, resource-intensive lighting calculation on the host computer before compiling the game. The output is one or more textures containing shadows and colour bleeding between adjacent objects from bounced light. Affected objects will refer to these textures when rendered.

To minimise the number of triangles to be rasterised by the headset's graphics processor, the surrounding stadium is made up of a 360-degree image projected to the inside of a sphere that is slightly larger than the football pitch. The sphere is made up of far fewer triangles than would a corresponding 3D model. The sky portion of the 360-degree image is cut out to allow the pitch to be lit from the virtual sun while having the stadium cast plausible shadows onto the pitch.

In addition, smaller visual details like wrinkles in clothing are achieved with normal maps as opposed to using highly detailed geometry. A normal map (see Fig. 16 is a texture that is mapped onto a mesh using UV coordinates, just like a colour texture. However, its texels' R, G, and B components denote the 3D normal direction at the corresponding points on the faces of the mapped mesh. When rendering a mesh with a normal map applied to it, pixels may reflect light differently, or be darkened as if in shadow despite not being blocked by

actual geometry. The normal map applied to all players is adapted from one contained in the supplementary material for the Unity game engine on the SMPL website [29], and is shown in Fig. 17.

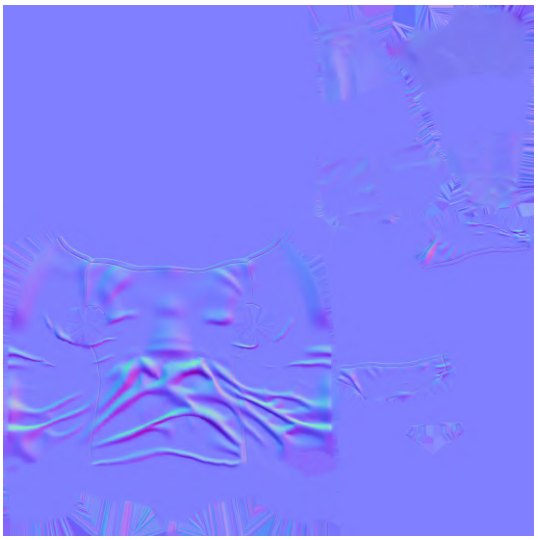


Fig. 16. Normal map containing detailed surface information, such as wrinkles in clothing.

(a) Plain SMPL mesh without any textures.

(b) Normal map applied to SMPL mesh, imitating a detailed, wrinkled surface.

Fig. 17. Normal map texture (left) and as applied to an SMPL mesh (right).

The player models also have ambient occlusion textures applied. These black-and-white textures darken areas of the mesh that receive little light due to self-occlusion, such as crevices. These textures were hand-drawn by darkening the boundaries between items of clothing, such as the parts of the legs just below the shorts, giving the illusion of the clothing having thickness despite being painted onto a smooth surface. Like albedo and normal textures, ambient occlusion textures are mapped to the mesh using UV coordinates.

Lastly, textures are also used in favour of geometry for the goal models. Instead of modelling the nets using 3D geometry, they are flat surfaces with a semi-transparent grid texture.

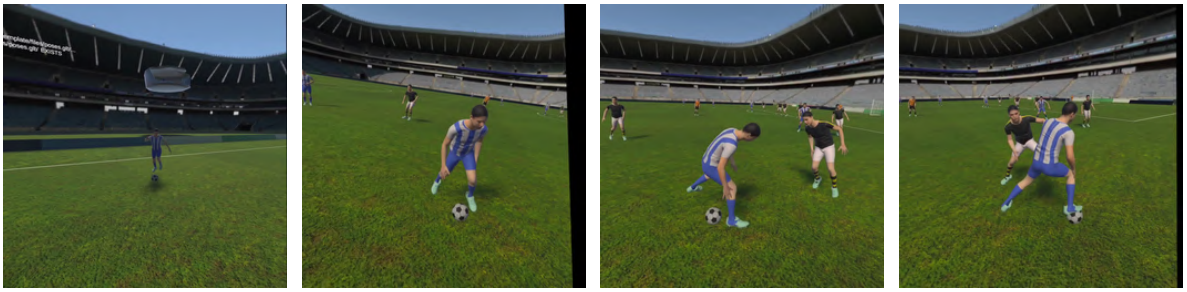
4.4 The Reconstructed Football Match in Images

In this section, the results from the entire project, including the contributions from the two other master's theses, will be presented through images. In Fig. 13, the starting room is

displayed where the user begins and can interact with the VR environment. Here, the user can play the football sequence, rewind and fast-forward, and descend onto the football pitch by pointing the controller's beam at the miniature football pitch and pressing the controller's trigger button with their index finger.

Figure 18 illustrates a sequence of images where the user jumps onto the pitch, positions themselves next to the players, and observes the game situation from an outsider's perspective while standing on the field. The tactical board is also visible in their left hand.

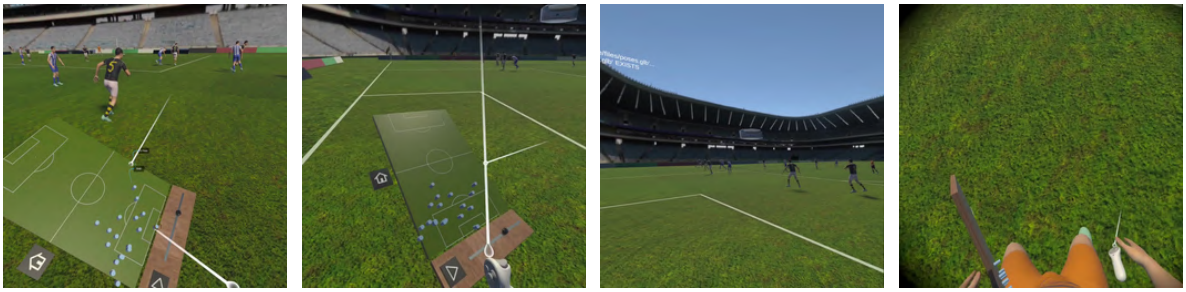
(a) (b) (c) (d)



(e) (f) (g) (h)

Fig. 18. The sequence of images illustrates how the user interacts with the VR environment. In (a), the user is standing in the starting room, pointing at the pitch to teleport down. In (b), the user has teleported and is now on the pitch with the tactical board in front of them, and in (c), the player presses the "play button" to start the game. In the subsequent (d), (e), (f), (g), and (h) images, we observe the user standing still and watching the game from an outside perspective (as if they were another player on the pitch).

In Fig. 19, the user navigates into a player using the tactics board. The user takes the right-hand controller and directs the beam at the goalkeeper on the tactics board, then becomes the goalkeeper by pressing the controller's trigger button. The user then observes the game in the subsequent frames from the goalkeeper's point of view.



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

Fig. 19. The sequence of images illustrates how the user interacts with the VR environment. In (a), the user stands still on the field and points at the goalkeeper to become him. In (b), the user has become the goalkeeper, which is further emphasized in (c). In the final image (d), the user is fully integrated with the goalkeeper, following the game as the goalkeeper did during the match.

5

Estimation of Player Colour Textures

This chapter details the development, application, and subsequent improvements made to the texture generation model for estimating player textures using differentiable rendering. The structure is somewhat unconventional, due to the nature of our methodology. The chapter is structured as follows.

First, the preliminary setup, the architecture, and its functionality are introduced in Section 5.1. The section describes the process of extracting target patches by creating and linearly interpolating bounding boxes around tracked players to a standard size of 256×256 pixels. Next, 3D models of players are generated from pose estimates using SMPL parameters, and starting UV texture maps are applied. Finally, in the training loop, the textures are iteratively refined by rendering each player's pose, comparing the rendered image to the target patch, and updating the texture in proportion to the difference.

Then, in Section 5.2, initial results are produced and evaluated to identify the primary areas of improvement. The section reveals issues including high-frequency noise, blurry transitions between regions of different colour, and non-updated pixels. These problems result in unwanted colours, unclear boundaries, and incomplete textures for areas not captured by the camera. This is especially problematic for more static players such as goalkeepers and referees.

After that, the reasons behind these errors are explored in Section 5.3 to gain a deeper understanding of the problems and identify potential solutions. The section describes why the poor texture quality is due to image-related issues, inaccuracies from previous work, and match-related factors. The image-related issues include low patch resolution and "sparse sampling," while previous inaccuracies involve misidentified player IDs and inaccurate pose estimates, and the match-related factors include player occlusions, fluttering clothing, and varying lighting conditions, leading to noise and blurry transitions between colours.

Finally, Section 5.4 presents and evaluates a variety of solutions that were explored to address these problems: first, how creating a colour palette and enhancing it helped address dull colours; then, how discretization of colours, low-dimensional parameterisation, and perceptive loss, improved texture quality; and finally, how diffusion models were integrated to inpaint missing texture areas and refine details, although challenges remain with maintaining pose accuracy and generating realistic textures.

5.1 Constructing the Differentiable Rendering Pipeline

The texture generation pipeline serves as a testbed for evaluating different approaches to the extraction of player appearances. As mentioned in Section 3.3, the rendering pipeline has five main input files as described in Table II. Additionally, there is an `.obj` file obtained from the SMPL website [29], defining the faces of the mesh and the UV coordinates of its vertices.

TABLE II. Inputs to the rendering pipeline

Input File	**Data Format**
Frames of the video clip	`.png`
Estimated poses	`.pkl`
Tracking data	`.pkl`
Camera parameters	`.pkl`
UV texture map as a starting texture	`.png`

5.1.1 Extracting Target Patches

The tracked players are provided in the form of bounding boxes, as seen in Fig. 20. For each player ($p \in P$) and each frame ($f \in F$), an array describes the bounding box according to center$_x$, center$_y$, width, and height (denoted as $b \in [0, W] \times [0, H] \times [0, W] \times [0, H]$). That is, the combined shape of all bounding boxes is $|P| \times |F| \times 4$. These bounding boxes are used to create patches centred around the players. The patches are cropped from the frame with the help of b, however, sometimes the dimension of the patch is smaller than the target shape of 256×256 (never larger due to the resolution of the frame which is full HD). Then, the images are upscaled using the linear interpolation from OpenCV library [47]. Linear interpolation estimates unknown pixels' values by blending the colour between known pixels. It calculates the intermediate value by assuming a linear change between these pixels, based on their distances and the values they hold, i.e., the images are upscaled from the original size to 256×256 as visualised in Fig. 21. The patches can also be referred to as targets, denoted as $y_{p,f} \in Y$ where $Y \in [0, 1]^{256 \times 256}$, and can be seen in Fig. 22.

Fig. 20. Visualization of the bounding boxes estimated by the first work on a match between IFK Göteborg and Norrköping.

(a) (b) (c)

Fig. 21. (a) the visualised bounding box of a player; (b) the extracted target patch of the player using the bounding box; (c) the resized patch after linear interpolation.

Fig. 22. Three different players (of which the right is the referee) from the same camera angle in different frames.

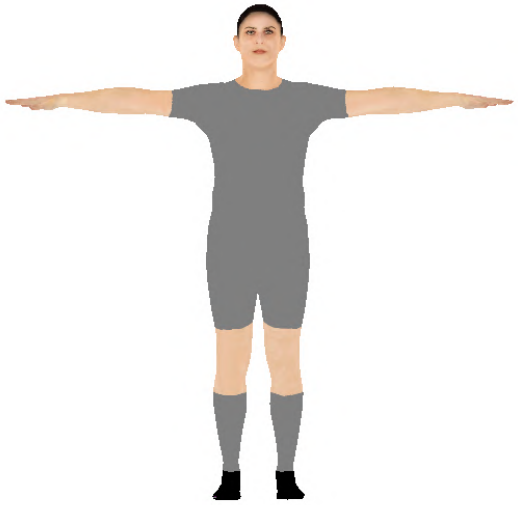
5.1.2 Building Meshes from Provided Pose Sequences

The second work provides estimates of the poses of all players and the location of the camera. Importantly, the camera location is expressed as an offset from each player, when pointed directly at them, meaning there is no single, global camera location. These parameters are thus not the same as the global ones used to orient the camera in relation to the pitch in Chapter 4.

The poses are in SMPL format, each containing the players' body pose $\vec{\theta}$, their body shape $\vec{\beta}$, and their orientation. These parameters are sent through the SMPL-X Python package [45] to generate posed 3D models consisting of a set of vertex positions, face information, alongside some auxiliary information. However, as the faces and other parameters remain constant regardless of the pose, the vertices are the only interesting parameter which is used to render the posed meshes later. The camera translations $\vec{t} \in \mathbb{R}^{|P|\times 3}$ are later used to align the virtual camera with the perspective of the physical camera.

Lastly, each player is assigned their own starting UV texture map $T \in [0,1]^{S\times S\times 3}$, where S is the side length of the texture in texels, which is to be updated in the optimisation process. All players start with the same texture that is shown in Fig. 23, which is mapped to the mesh using the vertices' UV coordinates before rendering.

(a) Starting UV texture map for each player.



(b) Front view of the texture mapped to an SMPL model.

Fig. 23. Visualisation of the SMPL UV texture map applied on the SMPL 3D model rendered in a T-pose with PyTorch3D.

5.1.3 Generating Posed Clothing Masks

The final step of pre-processing is to render all the meshes in their estimated poses, with a special mask texture applied for extracting only pixels belonging to any piece of clothing. We call these the *posed clothing masks*, of which there is one for every player in every frame. These are later used to mask out pixels of the rendered images for which to calculate the loss.

Before any mesh can be rendered, some initial parameters are set. The first is the specification

of the subset of players whose textures are to be generated; updating all player textures in unison requires more memory than is available. Additionally, a virtual camera is set up to define the viewpoint of the scene. This involves configuring a perspective camera with the provided camera calibration parameters, including its estimated position and focal length f'. The focal length is set to $f' = 5000$, stemming from the assumption of a weak perspective camera model used in the results from the second work. The weak perspective camera model is a simplification where depth variation is considered small compared to the distance from the camera, allowing for simpler computations while maintaining reasonable accuracy.

The renderer also incorporates a rasterisation setting, which dictates how the projected scene geometry is converted into fragments on the screen. This includes defining the resolution of the output image ($W \times H$), which is set to 256×256 to match the target patches. Additionally, the number of faces to aggregate depth-wise and the blur radius are set. The optimisation process will only update the textures of the rendered meshes, not their geometry. Furthermore, the scene is very simple, containing only one mesh at a time. Consequently, the faces to aggregate per pixel is set to $K = 1$ and the blur radius is set to 0 to mimic a conventional rasteriser. To light the scene, an ambient lighting model is used. This provides a consistent, uniform light on the entire scene.

Finally, the shading model is defined using a soft Phong shader [48], which is responsible for calculating the colour of every rendered pixel based on the posed mesh, its texture, and the lighting. The camera translation $\vec{t_z}$ is used to move the virtual camera so that the rendered mesh aligns with the target patch exactly.

For each player in every frame, the scene is configured as outlined above, the clothing mask texture is applied, and the textured mesh is rendered to produce the posed clothing mask. See Fig. 24 for an example.

(a) Target patch.

(b) Posed clothing mask overlaid onto the target patch.

Fig. 24. Visualisation of the alignment of a posed clothing mask.

5.1.4 Training Loop of the Rendering Pipeline

All approaches that require iterative refinement were run through the training loop of the pipeline. For each iteration of this loop and every player, it selects one of the frames where they were visible in the video clip. The corresponding pose $\vec{\theta}$, shape $\vec{\beta}$, and camera translation are then fetched, and the player's texture is rendered in the translated camera on their generated SMPL mesh. The rendered image is compared to the target patch using one or more loss functions. The primary one is mean absolute error (MAE), or ℓ_1 loss, applied to each pixel belonging to the player's kit, utilising the posed clothing masks. (See Section 5.4.4 for further details on loss functions.) Finally, the gradients of the total loss are propagated through the rendering pipeline back to the texture. The texels are updated using Stochastic Gradient Descent (SGD) without momentum.

5.2 Observing Poor Textures

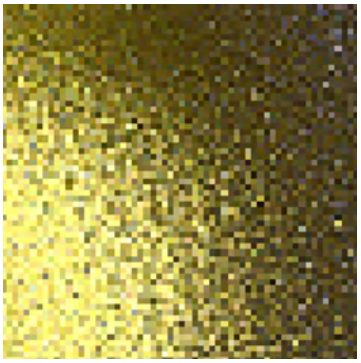
By running the pipeline once without any post-processing, the results shown in Fig. 25 can be observed. These results indicate that the texture can be rendered through differential rendering; however, several issues contribute to the poor quality of the texture. In this section, the problems associated with the generated textures will be discussed.

Fig. 25. Visualisation of the textures applied on the player characters rendered with PyTorch3D. At the top is the kit of IFK Göteborg, followed beneath by an AIK player, succeeded by their goalkeeper, and finally, the referee is displayed.

The first problem that can be observed is the high-frequency noise in the estimated UV texture map, as in Fig. 26a. The issue arising from high-frequency noise in textures concerns areas where a predominant colour should be present, yet other colours intrude. For instance, the previously mentioned area in the figure should exhibit a uniform yellow; however, shades of green, grey, and brown infiltrate this region. This phenomenon means that in a colour space,

these colours often appear at significantly distant locations (wide pixel spacing). Consequently, it becomes challenging to mathematically determine which pixels belong to specific areas. This also causes the overall colour to be perceived as somewhat dull and grey.

(a) High-frequency noise in the UV texture map.



(b) Blurry transitions in the UV texture map.

(c) Non-updated texels, illustrated with a rendered player.

Fig. 26. Problems observed in the raw output from the differentiable rendering model.

Another issue present is that the transitions between different regions of colour are not sufficiently sharp to be perceived as a high-quality estimated texture, as seen in Fig. 26b. This is evident when the colour shifts from yellow to black, resulting in a gradual transition between the colours. Ideally, the transition should be as sharp as seen on an actual kit.

The last issue with the estimated textures is the presence of non-updated texels. This indicates that certain texels have never been visible in any frame or pose of the player. As seen in Fig. 26c, the goalkeeper is shown from two distinct perspectives. The front side, facing the camera, has been rendered with a texture closely resembling the actual football kit. In contrast, the back side, which faces away from the camera towards the goal and remains unseen, exhibits texels that are non-updated and thus retain the starting colour. Similarly, the referee often runs with only one side facing the camera, as a short match sequence typically does not include multiple changes of direction, thus reducing the need for the referee to rotate. When a player's body parts are not visible, it results in non-updated texels.

5.3 Understanding Causes of the Poor Textures

In the previous section, three main categories contributing to poor texture were identified: high-frequency noise, blurry colours, and non-updated pixels. This section will delve into the origins of these issues, categorising them into three areas: image-related causes, results from previous works, and factors beyond control that arise from match situations.

5.3.1 Image-Related Causes

The primary challenge causing significant problems throughout the texture generation process is the resolution of the patches. The original patch from a frame can vary in size depending on the player's position on the pitch; a player closer to the camera will be composed of more pixels than one farther away. To make the pipeline agnostic to any resolution of

patches, all patches are linearly interpolated to a size of 256×256 pixels. However, this does not eliminate the fundamental issue of low patch resolution, resulting in lower-quality textures for players far away compared to those closer to the camera.

The second issue is dubbed "sparse sampling," which arises from the discrepancy in resolution between the texture and the rendered image. When rendering an object with a high-resolution texture onto a low-resolution image, only a small subset of all texels on surfaces facing the camera will be sampled in the rasterisation process.

Let $T(u, v)$ denote the texture map function for sampling from the texture at UV coordinates (u, v). The rendered image $I(x, y)$ samples the texture at discrete intervals determined by the image resolution, the focal length of the virtual camera, and the distance of the object from the camera. The relationship can be described as in Eq. (27):

$$I(x, y) = T(u(x, y), v(x, y)), \tag{27}$$

where $u(x, y)$ and $v(x, y)$ map pixel coordinates to the corresponding UV coordinates. If the sampling resolution is not sufficiently high, many texels are skipped between x and $x + 1$, resulting in a sparse sampling of the texture. As a consequence, this may lead to some texels not being assigned any colour, or noisy textures where neighbouring texels are optimised from separate frames. Fig. 27 illustrates how higher resolution allows for more detailed textures, but exacerbates the issues caused by sparse sampling.

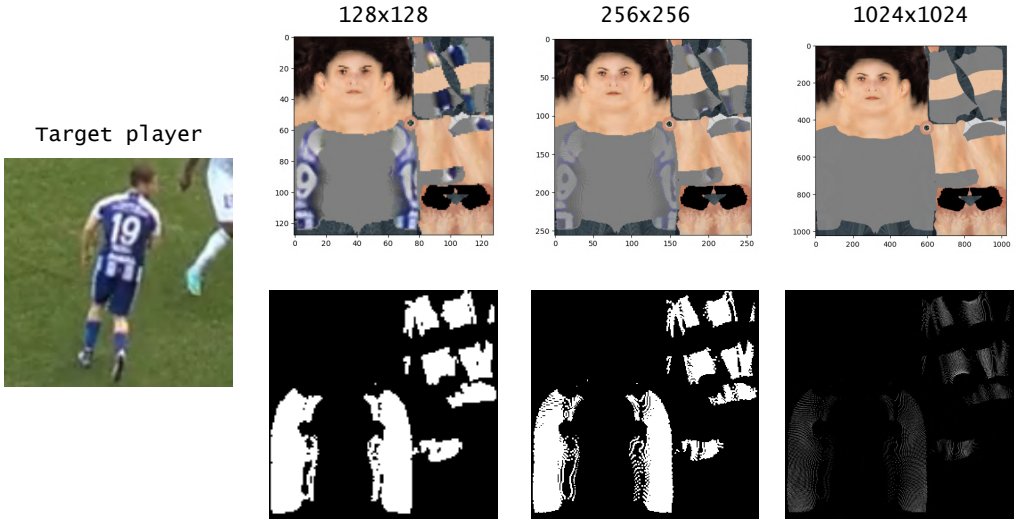


Fig. 27. Visualisation of sparse sampling from an optimisation run on one view for 100 iterations. The target patch is shown on the far left. Next, three different sizes of UV texture maps are visualised, illustrating the consequences of high-resolution textures. Nearly the entire back of the player is estimated at both 128×128 and 256×256, whereas only a subset of texels are updated for the 1024×1024-texel texture. In the lower row, updated pixels are shown in white, and non-updated pixels are shown in black.

5.3.2 Results from Previous Works

The second category of causes for the issues listed in the previous section is related to the results from the other theses. These are causes based on their results, which serve as input to our model. Consequently, if their results are inaccurate, it leads to inaccurate results in our model as well. Primarily, there are two main causes: the first is the changing of player IDs, and the second is poorly estimated player poses. The latter issue can also stem from poorly estimated bounding boxes, but this will not be discussed in this thesis, as it rather is an issue that the second work has to deal with.

The first issue is the exchange of IDs between players. This occurs when multiple players have been tracked and assigned IDs that can be followed across multiple patches, but in a specific frame, their IDs are swapped, usually due to player collisions or occlusions from the camera's perspective. Consequently, the players end up with each other's IDs as seen in Fig. 28. In the figure, the player ID is attached to one player in the first patch, but suddenly switches when the two players come into contact. This can also occur between players of the same team, which is more difficult to detect as the image resolution makes it hard to distinguish between players wearing the same kit.

Fig. 28. Visualisation of two players switching IDs during four frames. The switch occurs between an AIK and an IFK Göteborg player.

As a consequence of the switch, the colours updated into the texture from the initial patches will not be reflected in the following patches due to the change in the colour palette between players. Additionally, the estimated poses in these scenarios are often of very low quality, as evident in Fig. 28. Both of these may contribute to the high-frequency noise observed in the preliminary textures.

Another scenario is when a player moves out of frame and then reappears. Currently, the player is assigned a new ID when reappearing. Fortunately, this is less problematic because it simply results in two separate textures being generated. However, each of those textures will be optimised using only the subset of patches with the same ID, meaning they are likely to be less detailed than a single texture optimised on both sets of patches.

The second cause of issues from previous works is poorly estimated poses of the players. This occurs when the estimated player pose does not align perfectly with their actual pose in the patch. Note that almost no poses are perfectly estimated, and there is always some error, typically for extremities like the feet, hands, and head. However, in more extreme cases, the entire body is inaccurately estimated, as seen in patches 2 and 3 on the top row, and patches 1 and 2 on the bottom row in Fig. 28.

The issue is clear: to achieve an accurate estimation of a player's texture, the estimated pose must align precisely with the player's pose in the patch. Any misalignment can result in incorrect colours being incorporated into the estimated texture, such as the opponent's kit or the grass from the pitch, leading to high-frequency noise. Additionally, even slight inaccuracies in pose estimation can cause patterns, such as those on jerseys, to misalign. This misalignment can result in blurred transitions between regions of different colour, affecting the overall quality of the estimated texture.

5.3.3 Inevitable Match-Related Problems

The third and final category of causes for the issues arising with the estimated textures involves factors beyond control that stem from match conditions. This includes situations such as players obstructing each other (from the camera's perspective), fluttering clothing on the players, variations in lighting between images (for instance, when a player runs into the shadow cast by the stadium roof), and, lastly, players being obscured from certain angles during a sequence of the match.

(a) Two players in a patch, with the player in the foreground covering the pose-estimated player in the background.

(b) A player with a correctly estimated pose but with a fluttering jersey.

(c) Showing the problem with different light conditions within a small area and its effect on the player's textures.

Fig. 29. Showcasing three different inevitable match-related problems.

The first match scenario that often causes issues in the model is when players occlude each other, as seen in Fig. 29a. In the figure, pose estimation has been performed on the player in the background. However, an opponent has moved into the frame (from the camera's perspective), causing certain body parts of the pose-estimated player to be obscured. The model used by the second work for pose estimation does not account for the depth of players relative to each other in the image. Consequently, when the estimated player model is overlaid

on the target patch, it is placed on top of both the background and foreground players. This results in colour leakage from the foreground player's body parts onto the background player during pixel comparison between the texture applied to the player model and the patch.

The second scenario involves the players' clothing fluttering, as illustrated in Fig. 29b. In the figure, the player is accurately estimated, but when the player is aligned onto the patch, there is still a discrepancy between the player model and the jersey. This discrepancy arises because the jersey in the figure flutters due to the match situation where the player is running and making a quick turn. The consequence of this is that when the texture is compared to the target patch, the clothing (the jersey in this case) does not match, which causes a shift that introduces new colours where they do not belong. This can lead to blurry transitions between, for example, stripes on the jersey, as shown in Fig. 29b.

The third scenario involves how varying lighting conditions affect the colour of players, as seen in Fig. 29c. The figure shows that players standing in brightly lit areas appear significantly overexposed, often rendering the entire player as white. In contrast, a player standing just one metre away in the shade displays perfect colour rendering. This scenario becomes particularly relevant when tracking a player across multiple target patches, as the player may move in and out of these bright areas, resulting in dramatically different colour renderings. As colours shift between patches, new colours are introduced based on the player's position on the pitch. Consequently, within regions of the same colour, other colours (often overexposed whites) can seep in, introducing high-frequency noise.

5.4 Solutions to the Poor Textures

In the preceding sections, the issues related to the textures were initially examined, and the underlying causes of these problems were discussed. In this section, potential solutions to the problems, based on these causes, will be explored.

5.4.1 Extracting a Colour Palette

The initial solution is not a direct measure for a specific problem, but rather a preliminary step that will be utilised in several subsequent solutions. Our objective in creating a colour palette is to determine, based on the players' kits from the target patch, the primary colours present in the clothes. Additionally, the aim to verify whether these colours correspond to the true colours the kits are supposed to have or not.

The initial step in identifying the colour palette from the players' kits involves isolating the clothing from the target patches. This is accomplished using the posed clothing masks that were produced in Section 5.1.3. By generating a mask for each patch, we can extract only the clothing from the patches, as shown in Fig. 30. Once these new masked patches are produced, K-means clustering is applied to all the pixels in the texture to produce a colour palette featuring the five most dominant colours, as shown in Fig. 31a.

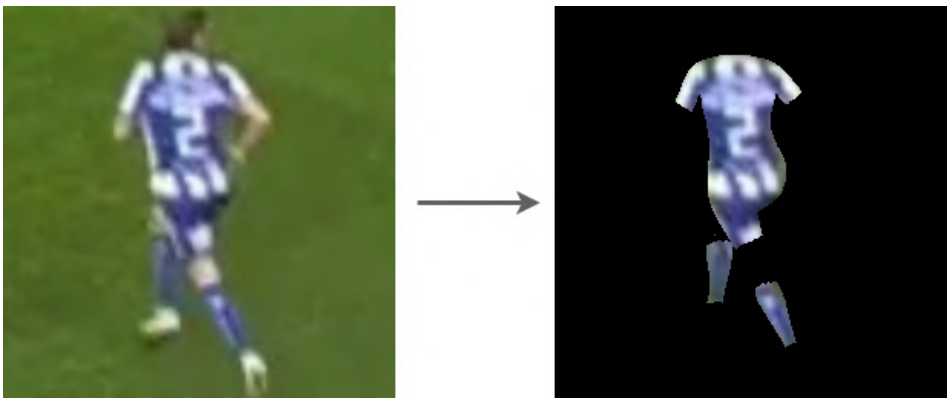


Fig. 30. Illustration of how a target patch is masked with the pose clothing mask on the player.

(a) The five most dominant colours observed in the masked patches (clothing data is taken from IFK Göteborg's player kit).

(b) The two true colours from the IFK Göteborg player's kit.

Fig. 31. Comparison between the colour palette observed on the target patches and the true colours on the IFK kit.

As observed in the colour palette in Fig. 31a, the colours are quite dull and greyish. Furthermore, there is no white colour present, unlike the true colour palette in Fig. 31b. This suggests that the colours represented in Fig. 30 are not highly saturated, but rather quite the opposite: they are weakly saturated and low in brightness (low value). Therefore, an attempt was made to create a more distinct colour palette by segmenting the saturation-value colour space into regions and pushing the five most dominant colours to extreme points based on the region they fall into, while retaining their hue. This is visualised in Fig. 32, for an arbitrary colour.

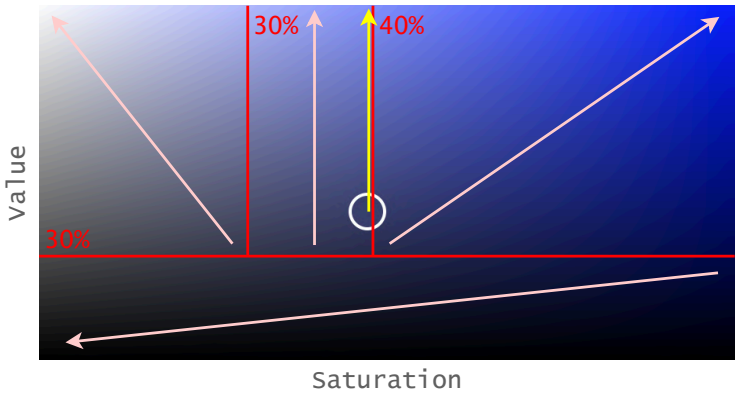
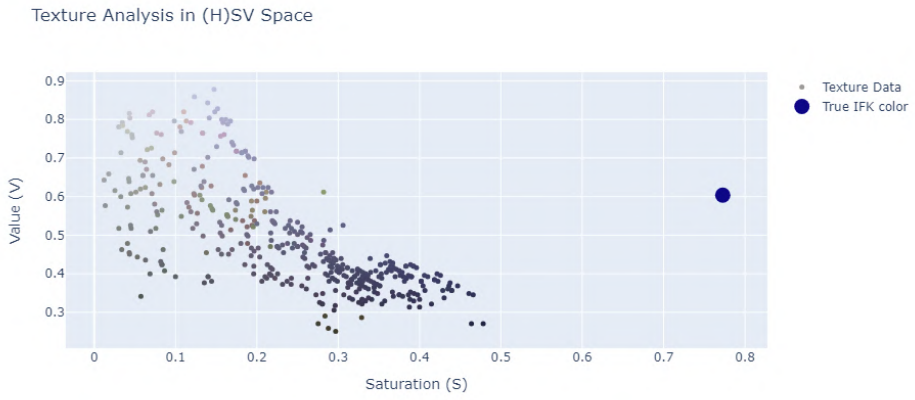


Fig. 32. The HSV colour space is divided into four regions. The light red arrows represent to which edge or corner colours within that region are pushed. The yellow arrow shows how the value of a colour falling into one of the regions is pushed toward an extreme.

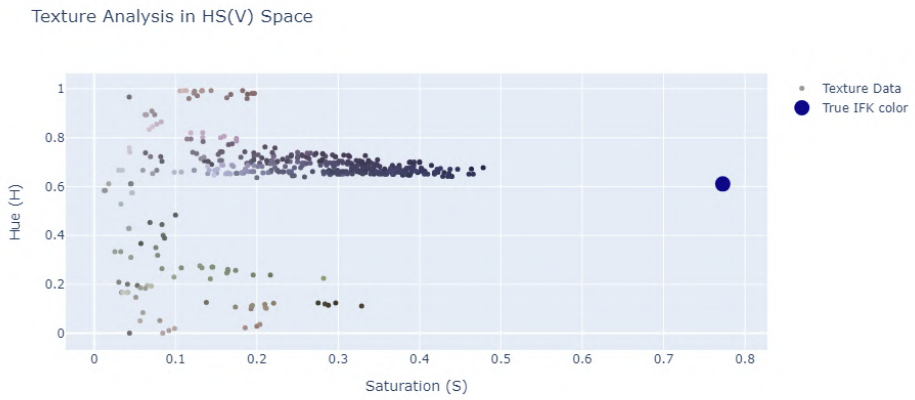
As illustrated in Figure 32, the colour space of saturation and value is divided into four segments. The purpose of these segments is to represent white (far left), black (bottom), colours with lower saturation but high value (centre), and colours with high saturation and value (far right). In each segment, colours are pushed towards the extremes to create more distinct and vivid colours. In the white colour zone, all colours are adjusted to zero saturation and a value of 100. In the black colour zone, both saturation and value are reduced to zero. In the central colour zone, the value is increased to 100 while retaining the original saturation. Finally, in the last colour zone, both saturation and value are set to 100. Note that all colours keep their hue.

The rationale for dividing the colour zones is based on several considerations. Firstly, the use of four distinct zones is justified by the fact that all lighter colours can be represented by white. The whiter the texture colour, the clearer and better estimated the texture appears. Similarly, this logic applies to all dark or black colours, where they can be represented as solid black. The middle colour zone is designed to capture colours that are lower in saturation but still distinct on a uniform. Not all colours are fully saturated and are often perceived as more aesthetically pleasing with just an increase in brightness, i.e., value. The final zone is intended to capture and enhance slightly muted colours. Colours with a saturation above 40% are perceived better with increased saturation in a texture.

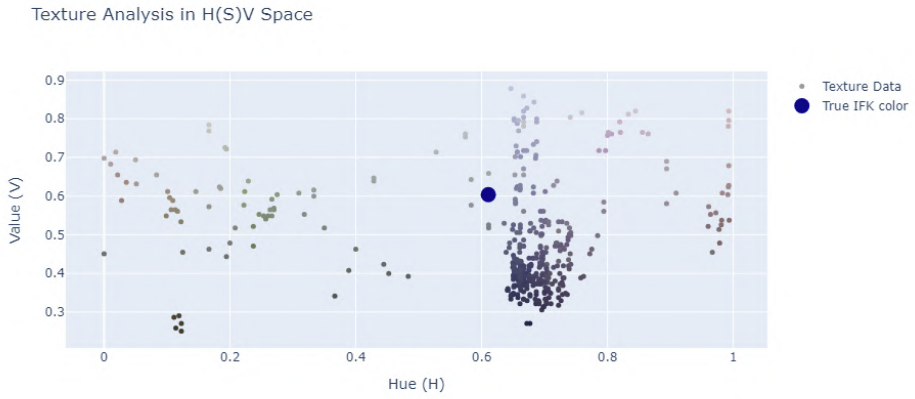
We explore this relationship further by measuring the colours of hand-picked pixels in the target patches and comparing them with the known colour. Specifically, target patches belonging to the same IFK player are sampled at locations whose UV coordinates are known to be blue on the desired texture. The findings are presented in Figs. 33a-33c where the measurements' hue, saturation, and value (brightness) components are plotted against each other.



(a) Plot where the x-axis represents *saturation*, while the y-axis represents *value*.



(b) Plot where the x-axis represents *saturation*, while the y-axis represents *hue*.



(c) Plot where the x-axis represents *hue*, while the y-axis represents *value*.

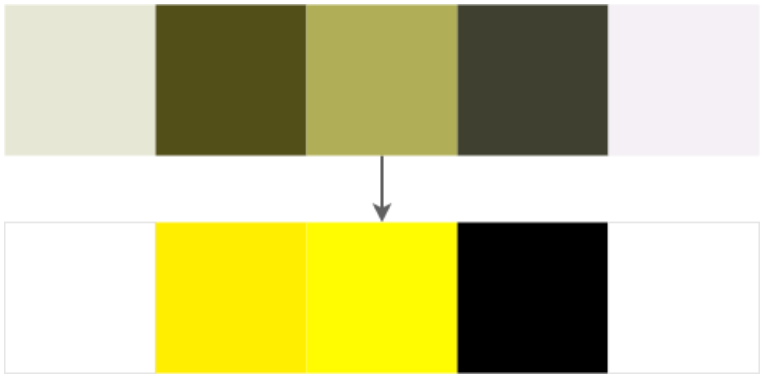
Fig. 33. Relationship between hue, saturation and value in the IFK Göteborg football kit. The data points are plotted to show the distribution of pixels of a player. The larger data point illustrates the true blue colour of IFK Göteborg. This plot provides insights into how the pixels are distributed compared to the true (target) colours.

The rationale behind selecting the 30% and 40% thresholds was based on the distribution

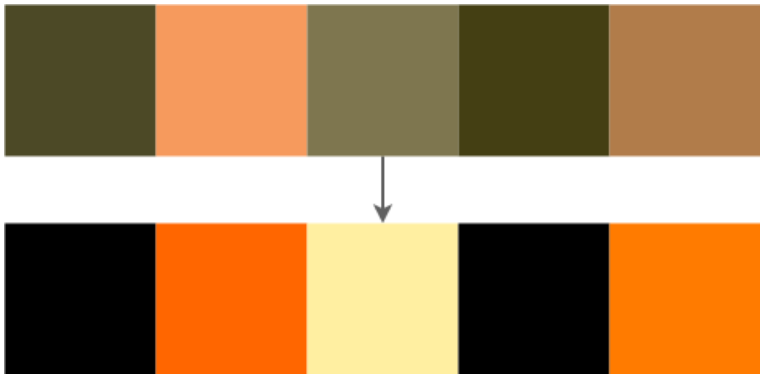
and the comparison between the observed pixels and the true colour. It was observed that the mean hue value was often accurate (as demonstrated by the use of K-means clustering; see Fig. 33b). However, the saturation values were consistently too low in the images, which can be attributed to a combination of the low resolution, the physical lighting conditions, as well as the interpolation involved in scaling up the images (as also seen in Fig. 33b). The decision to push the value to the extreme was based on observations, as the texture appeared more appealing with a higher value, even though this did not precisely represent the true colour.

By enhancing the colours in Fig. 31a through the saturation and value segmentation shown in Fig. 32, the colour palette presented in Fig. 34 can be achieved. When comparing this palette with the true palette in Fig. 31b, the similarity is evident. Additional examples are illustrated in Fig. 35. The difference is apparent. While some kits will not benefit from this colour segmentation, the majority of kits feature either black or white colours, often paired with a highly saturated colour, which is accurately reproduced using this method.

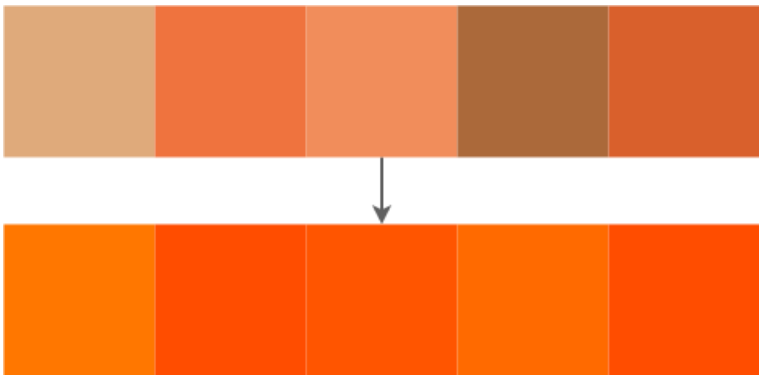
Fig. 34. Illustration of how the dull grey colour palette in Fig. 31a is shifted to colours with generally higher saturation and value by the colour segmentation in Fig. 32.



(a) Shift in AIK's kit colour palette from dull to vibrant.



(b) Shift in referee's kit colour palette from dull to vibrant.



(c) Shift in goalkeeper's kit colour palette from dull to vibrant.

Fig. 35. Illustration of the transition from the five most observed colours in the kits of AIK, the referee, and the goalkeeper to a more vibrant colour palette.

5.4.2 Discretisation of Colours

To address the first two observed issues, namely high-frequency noise and blurry textures, a discretisation method was considered. This method involves initially blurring the entire texture using a median filter, followed by assigning each resulting colour its closest counterpart in the extracted and post-processed colour palette. By first blurring the texture, the high-frequency noise within each region of the same colour can be reduced. The stronger the blur, the lower the high-frequency noise will be. However, this also introduces more blur, which is already a problem. Consequently, after making each colour cluster more uniform, the clustering process assigns a common colour to each texel within that cluster from the predetermined colour palette (as described in the previous section).

By applying a median filter to blur the texture, the aim is to eliminate noise within regions of the same colour and along the edges between different colour regions. The median filter replaces each texel's value with the median value in the neighborhood of that texel. The median filter is an algorithmic process rather than a traditional mathematical equation. It can be described as follows:

$$I_{filter}(x, y) = \text{median}(I(x + i, y + j)| - k \leq i, j \leq k), \tag{28}$$

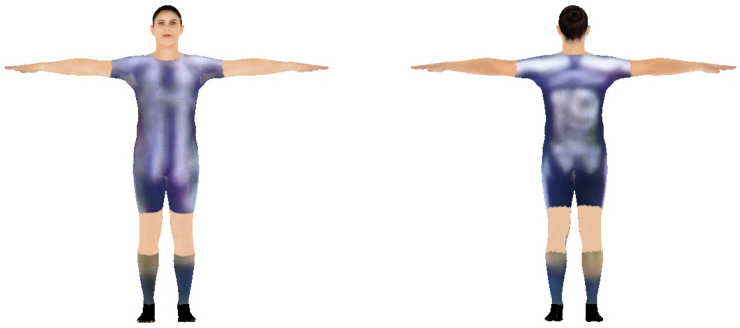
where $I(x, y)$ is the original image at pixel (x, y), $(x + i, y + j)$ represents the coordinates of the pixels within the neighbourhood centred at (x, y), and k defines the size of the neighbourhood (or kernel size).

After the texture has been processed to remove high-frequency noise using the filter, it becomes even more blurred than before. However, the colours should ideally become more uniform which can lead to more easily distinguished colours. Hence, the colour palette as described in the previous section can be utilised, where each pixel is assigned the values of the closest centroid by minimising the Euclidean distance as:

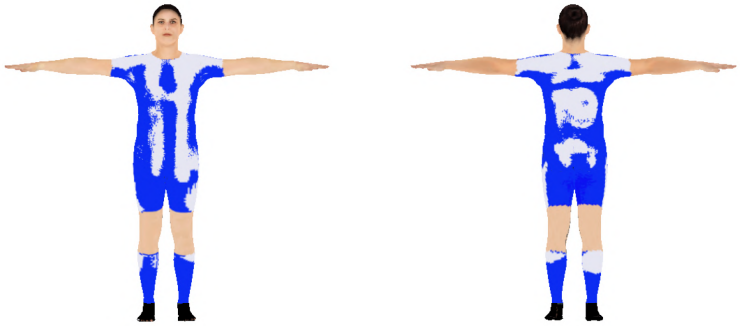
$$\text{Assign pixel } (x, y) \text{ to cluster } c_i \text{ where } c_i = \arg\min_i ||(h, s, v) - \mu_i||^2, \tag{29}$$

where (h, s, v) is the original colour of the pixel, and μ_i is the i-th centroid from the palette with colour (h_i, s_i, v_i).

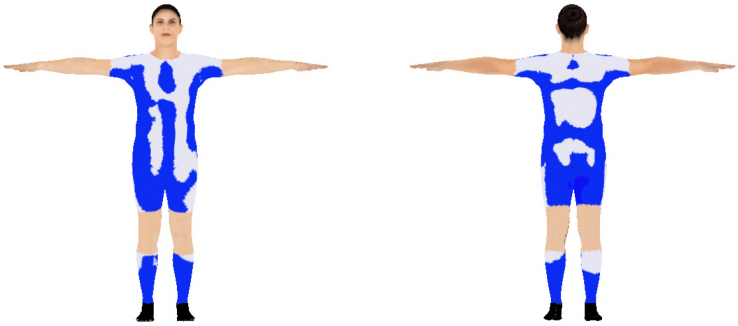
By initially blurring the texture and subsequently assigning each pixel to the nearest centroid according to the Euclidean distance, the following results can be achieved, as shown in Fig. 36.



(a) Raw rendered texture, estimated directly from the differentiable rendering model.



(b) Rendered texture, directly clustered from the raw texture without blur.



(c) Rendered texture, first blurred and then clustered from the raw texture.

Fig. 36. Outcomes from the discretisation with a rendered player in T-pose.

(a) UV texture map of the raw texture, estimated directly from the differentiable rendering model.

(b) UV texture map, directly clustered from the raw UV texture map without blur.

(c) UV texture map, first blurred and then clustered from the raw UV texture map.

Fig. 37. Outcomes from the discretisation in the UV texture map space.

5.4.3 Low-Dimensional Parameterisation

An appealing solution to the three problems of high-frequency noise in the texture, blurry transitions between regions of different colour, as well as hidden parts of players, is that of low-dimensional parameterisation. Instead of learning each texel individually, their colours are driven by a small set of learnt parameters in a one-to-many fashion. This drastically reduces the dimensionality of the solution space, and may, given a suitable parameterisation scheme, address multiple issues posed by updating the texels individually.

A simple parameterisation scheme is to learn a single colour for the jersey ($\mathbf{c}^{jersey} \in [0,1]^3$), the shorts ($\mathbf{c}^{shorts} \in [0,1]^3$), and the socks ($\mathbf{c}^{socks} \in [0,1]^3$), respectively. That is, the model consists of only three colours, each represented by a three-dimensional vector of the red, green, and blue colour channels. In combination with these clothing colour parameters are binary masks for the corresponding item of clothing in UV space: one marking all jersey texels ($M^{jersey} \in \{0,1\}^{S\times S}$), one for the shorts texels ($M^{shorts} \in \{0,1\}^{S\times S}$), and a third for the socks ($M^{socks} \in \{0,1\}^{S\times S}$).

At the start of each iteration in the training loop, the player texture is generated from the parameters:

$$T(i,j,k) = M^{jersey}(i,j)\cdot \mathbf{c}_k^{jersey} + M^{shorts}(i,j)\cdot \mathbf{c}_k^{shorts} + M^{socks}(i,j)\cdot \mathbf{c}_k^{socks}, \qquad (30)$$

where $i \in \{1,\ldots,S\}$, $j \in \{1,\ldots,S\}$, $k \in \{1,2,3\}$. The loss is then computed as before, by comparing the rendered image against the corresponding target patch, and the gradients are backpropagated through each texel and accumulated in the three colour parameters. The parameters are then updated before the next iteration.

Such an approach is fast to optimise due to the small parameter count; all parameters are updated in every iteration, as opposed to only the texels directly contributing to the rendered image. Consequently, pixels belonging to any piece of clothing will receive a plausible colour

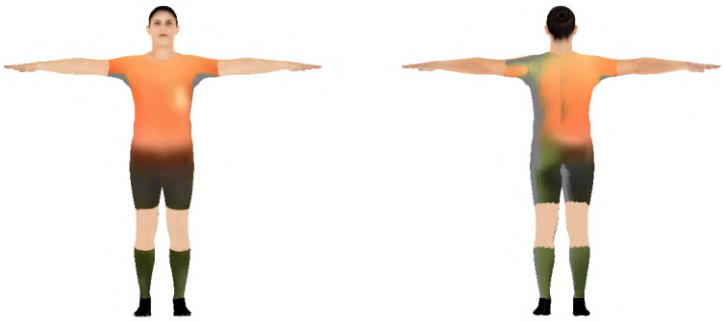
even if never in view of the camera. Additionally, the clothing masks directly define the sharpness of boundaries between items of clothing.

While it lacks expressiveness, and is incapable of producing details such as stripes, it is suitable as an initial pass before further texel-level refinement. This produces a best-of-both-worlds scenario, by adding detail only after ensuring that every texel has been assigned a starting colour that is likely to be close to its true colour, even if it was not visible in any frame. To illustrate the benefit of this, two runs are performed with and without an initial parameterisation phase, both optimising the texture of the referee whose left side is partially occluded throughout the clip. The first run executes $10\,000$ iterations of texel-level updates using ℓ_1 loss. The second run first executes 100 iterations of updates to the colour parameters to produce a starting texture, and then performs texel-level updates as in the first run. SGD is used in both cases, with a learning rate of 1.0 for colour parameters and 250 for texels. Fig. 38 shows the resulting textures directly, and Fig. 39 shows them when applied to posed SMPL models.

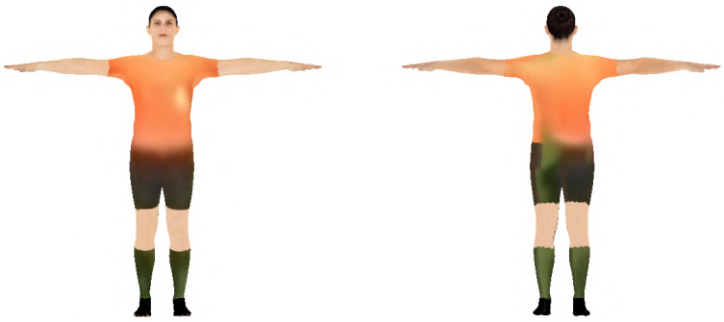
(a) Without parameterisation.

(b) With initial parameterisation.

Fig. 38. Comparison of textures (a) without and (b) with an initial parameterisation phase.



(a) Without parameterisation.



(b) With initial parameterisation.

Fig. 39. Comparison of textures (a) without and (b) with an initial parameterisation phase.

5.4.4 Exploring Different Loss Functions

The choice of loss function should be guided by insights about the nature of the problem and how training samples are produced. For example, the usage of mean squared error (MSE) assumes that all measurements belong to a symmetric normal distribution with the true value as its mean. This need not be the case in all scenarios. Indeed, as demonstrated in Fig. 31, the most prominent observed colours all tend to favour low saturation and brightness, indicating an asymmetrical deviation from the true colour.

Figs. 33b and 33c indicate that the measured hue is close to the actual one. However, the variance in both saturation and value is very high, with saturation being consistently too low and value nearly spanning the whole spectrum. This skew is likely partly caused by natural phenomena such as lighting conditions. The kit of a player in shadow will naturally be perceived as darker (lower value) than a player in direct sunlight. Conversely, in intense brightness, colours tend to desaturate (lower saturation), as shown in Fig. 29c in Section 5.3.3.

With an accurate representation of the location of the sun, the geometry of the arena, as well as the settings and physical properties of the camera, the transformation of colours from clothing as they are captured by the camera may be modelled in the differentiable rendering pipeline. Consequently, the rendered pixels of a player in shadow will be darker from not being hit by the virtual light source, thus closer to matching the true pixels of the target patch, resulting in little to no modification to the texture colours. While the introduction of such an accurate model is outside the scope of this thesis, a substitute that leverages the observations above is the use of an artificially shifted colour palette as described in Section 5.4.2.

Comparing Mean Absolute Error (MAE) to Mean Squared Error (MSE)

Two common element-wise loss functions for comparing the pixels of images are Mean Absolute Error (MAE or ℓ_1 loss) and Mean Squared Error (MSE or ℓ_2 loss). They are both special cases of ℓ_p norm loss functions, where $p = 1$ and $p = 2$, respectively. They are defined as

$$\text{MAE}(\hat{y}, y) = \frac{1}{WH} \sum_{i=1}^{W} \sum_{j=1}^{H} |\hat{y}_{ij} - y_{ij}|, \tag{31}$$

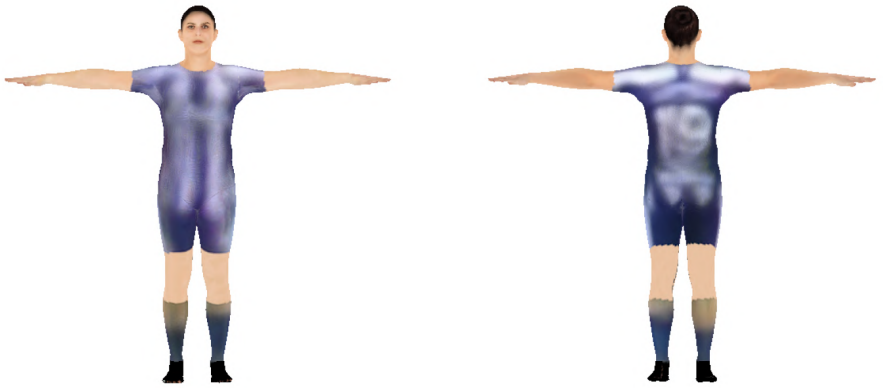
and

$$\text{MSE}(\hat{y}, y) = \frac{1}{WH} \sum_{i=1}^{W} \sum_{j=1}^{H} (\hat{y}_{ij} - y_{ij})^2, \tag{32}$$

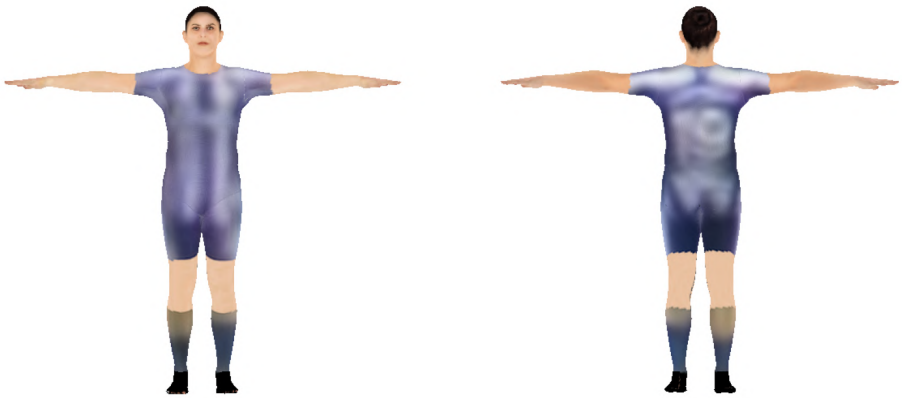
respectively, where $\hat{y}_{ij}$ is a prediction of the pixel y_{ij} from the ground-truth image, and W and H are the dimensions of both images.

In contrast to MSE, MAE is not differentiable at 0 as a consequence of the absolute value operation. However, this is typically managed by using any value from the sub-gradient of the absolute value at 0, which spans the interval $[-1, 1]$. Because it squares the residuals, MSE is more sensitive to large deviations between predictions and targets. The implication of this is that, in the presence of outliers in the data, the MSE loss minimum will be shifted more towards the average of these outliers. In the context of optimising image pixels, this can result in smooth, low-contrast images.

The two loss functions are evaluated qualitatively in identical scenarios. For the same IFK player, the initial parameterisation phase is run for 100 iterations, followed by $10\,000$ per-pixel updates using either ℓ_1 or ℓ_2 loss. SGD is used in both phases, with a learning rate of 1.0 for colour parameters and 250 for pixels.



(a) Texture optimised with MAE loss.



(b) Texture optimised with MSE loss.

Fig. 40. Illustration of the texture optimised with MAE and MSE loss.

The texture produced with MAE is slightly higher in contrast than that with MSE, which helps with legibility. MAE is thus kept as the primary loss function for penalising per-pixel deviations.

Perceptive Loss

A type of loss function that is less sensitive to the exact pixel values, and thus may be more robust against slight misalignment between the estimated pose and the player in different frames, is perceptive loss. It is based on the discovery that feeding two images through a pre-trained convolutional neural network and comparing their respective activations throughout the model aligns well with human judgement of their similarity [49]; when the ℓ_2 distance is small, humans tend to consider the images visually similar. Such a metric is particularly more sensitive to blurry inputs than older metrics like Structural Similarity Index (SSIM) [50].

We incorporate perceptive loss, in particular *Learned Perceptual Image Patch Similarity* (LPIPS) [49], into the texture generation pipeline in combination with the original per-pixel ℓ_1 loss. The LPIPS loss is applied to pairs of images consisting of (1) the rendered player

with its current texture, masked using the posed clothing mask, superimposed on top of the target patch, and (2) the corresponding target patch. See Fig. 41 for an example.

(a) Rendered image of the masked player with output from parameterisation phase, superimposed onto target patch.

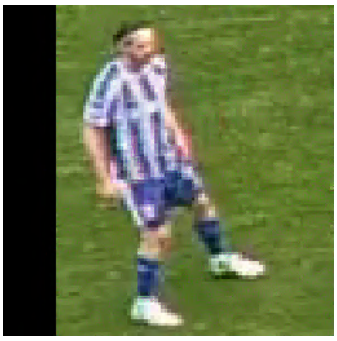
(b) Target patch.

Fig. 41. Example pair of images whose similarity is to be evaluated by the LPIPS metric.

The choice of the underlying model is a compromise between the quality of the results and the speed of training. To motivate the decision, both AlexNet [51] and VGG [52] are first evaluated in a test setup. In both cases, we start with the image in Fig. 41a and use the corresponding target patch in Fig. 41b as the target. The optimisation is run for $10\,000$ iterations using SGD with a learning rate of 10 and no momentum. The results are presented in Figs. 42 and 43. With AlexNet being a shallower network, it is also faster to evaluate. A second run of AlexNet optimisation is thus performed, using more iterations to closer match the wallclock time of the VGG network.

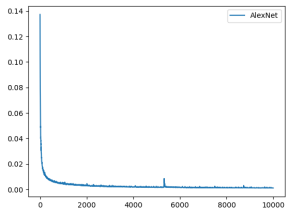
(a) AlexNet result (10k iterations).

(b) AlexNet result (52.4k iterations).

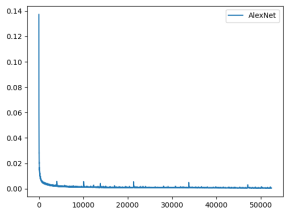


(c) VGG result (10k iterations).

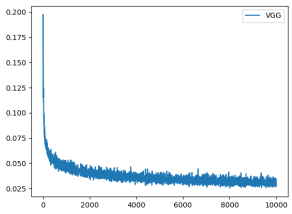
Fig. 42. Resulting images from the comparison between AlexNet and VGG backbone.



(a) AlexNet loss graph (10k iterations).



(b) AlexNet loss graph (52.4k iterations).

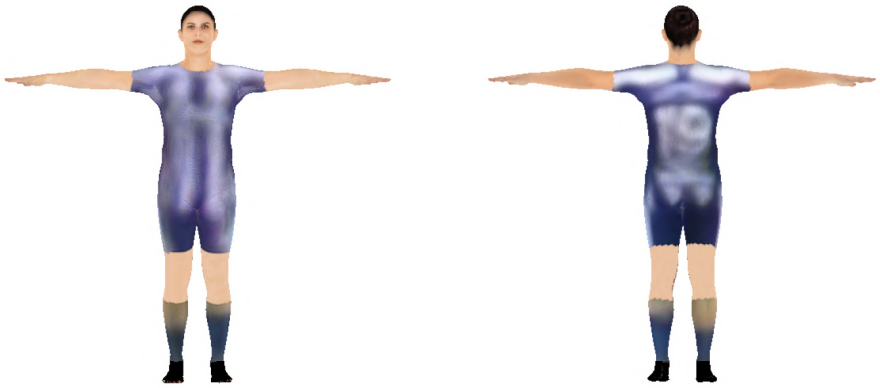


(c) VGG loss graph (10k iterations).

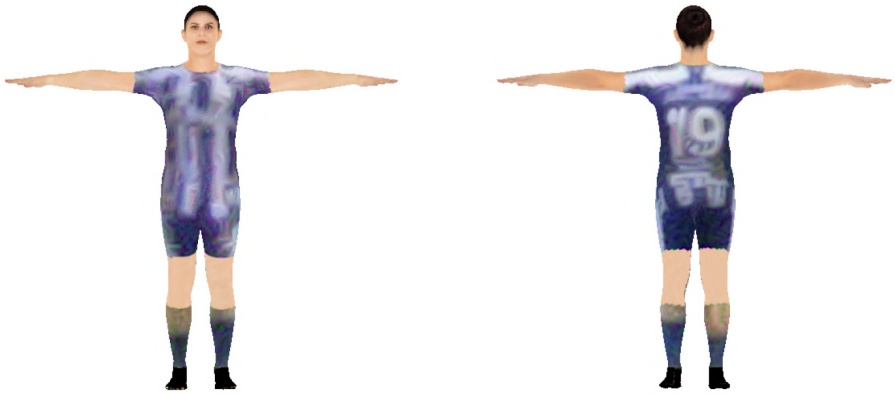
Fig. 43. Loss graphs from the comparison between AlexNet and VGG backbone.

The runs with AlexNet quickly converge to very low loss, but produce noisy outputs with little resemblance to details in the target image. In contrast, the VGG run produced a qualitatively superior image, and the reported loss appears to align better with the similarity between the images. The VGG network is thus chosen to be integrated into the texture generation pipeline.

In line with [17], we weigh the two losses to favour ℓ_1 loss 10-to-1. We run the optimisation process for $10\,000$ iterations on one of the IFK players with the joint objective, and then once more with only ℓ_1 loss as a control. For both runs, the learning rate is increased to 250 to account for texels receiving fewer updates due to not being visible in every frame. The texture resolution is decreased to 256×256 to increase the probability of texels being sampled from multiple views. The results are presented in Figs. 44 and 45.



(a) Texture optimised using ℓ_1 loss only.



(b) Texture optimised using ℓ_1 loss and LPIPS loss.

Fig. 44. Comparison of textures on posed characters.

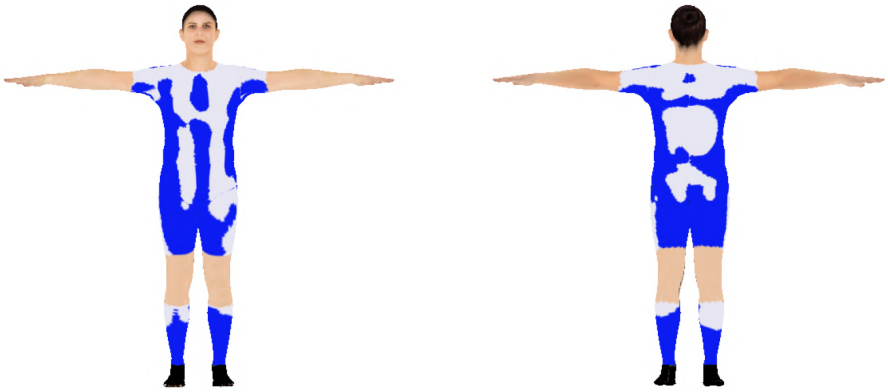
(a) Texture (ℓ_1 loss only).

(b) Texture (ℓ_1 and LPIPS loss).

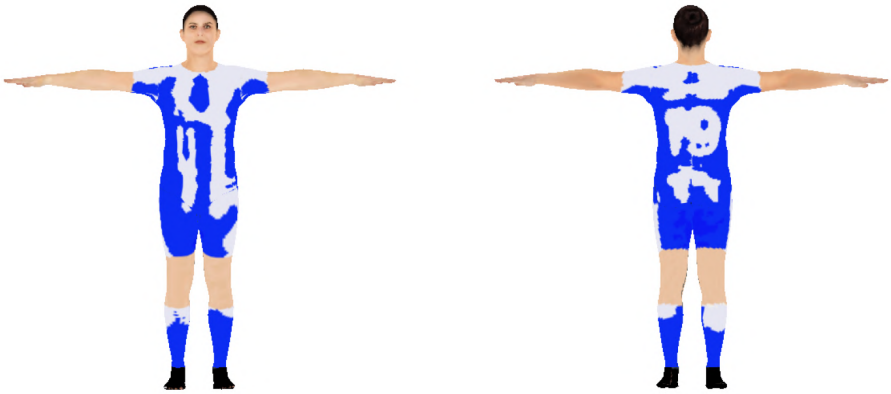
Fig. 45. Comparison of textures.

The addition of perceptive loss has two primary effects on the resulting texture. First, it accentuated details such as the jersey number, which is clearly legible in Fig. 44b but not as much in Fig. 44a. However, it also introduced artefacts in the form of swirls that make larger-scale details such as stripes less pronounced.

Finally, the colour discretisation step is applied to the two textures, the results of which are visualised in Fig. 46. While slightly less clear after post-processing, the jersey number is still more legible with LPIPS than using only ℓ_1 loss.



(a) ℓ_1 loss only, then discretise.



(b) ℓ_1 and LPIPS loss, then discretise.

Fig. 46. Comparison on posed characters after discretisation.

5.4.5 Utilisation of Diffusion Models

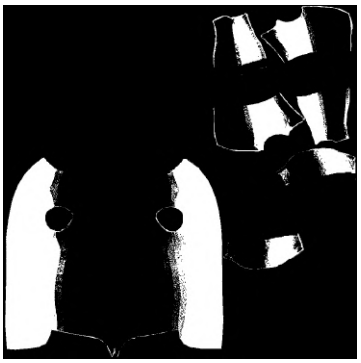
While previous works that apply denoising diffusion models to human texture reconstruction have been shown to produce high-quality results, our situation differs in two key aspects. First, most players are visible in many frames from multiple angles, and in optimal cases have rotated a full 360 degrees throughout the clip. Only utilising a single frame would leave out a lot of information. Secondly, while quantity is high, the quality of each frame is very low due to the low resolution of player patches. We thus explore the efficacy of revising pre-existing work to integrate with our texture generation pipeline.

Inpainting in Texture Space

We integrate the pre-trained SMPLitex model as a final step of the texture generation pipeline to address the problem with texels that were not observed in any frame, as an alternative to parameterisation with the potential of producing more detailed results. The pipeline is run with ℓ_1 loss, but the gradients of the texture are accumulated in a separate buffer to keep track of which texels have never been updated. At the end of the training, those gradients are used to produce a binary mask to separate between which pixels to keep and which to overwrite. Regions not belonging to clothing are excluded from the mask.

The generated partial texture map and the produced mask are fed into the SMPLitex algorithm with its default parameters. For the inpainting step, this includes 20 steps with a denoising strength of 0.8 and a guidance scale of 2.0, conditioned on the text prompt *"a sks [sic] texturemap of a soccer player*." The subsequent image-to-image step runs for the same number of steps and with the same guidance scale and prompt, but uses a much lower denoising strength of 0.05. Figs. 47 and 48 show the output generated from the partial textures of two players who were not completely visible throughout the clip. To enhance the stripes in the latter case, we post-process the IFK player's partial texture by applying the discretisation step to it before inpainting.

(a) Original texture for goalkeeper.

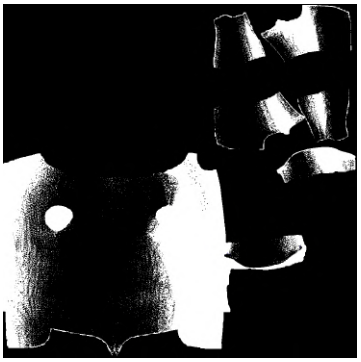


(b) Binary inpaint mask generated from accumulated gradients.

(c) Inpainted texture.

Fig. 47. Texture inpainting with SMPLitex on goalkeeper's partial texture.

(a) Original texture for IFK player (after discretisation).



(b) Binary inpaint mask generated from accumulated gradients.

(c) Inpainted texture.

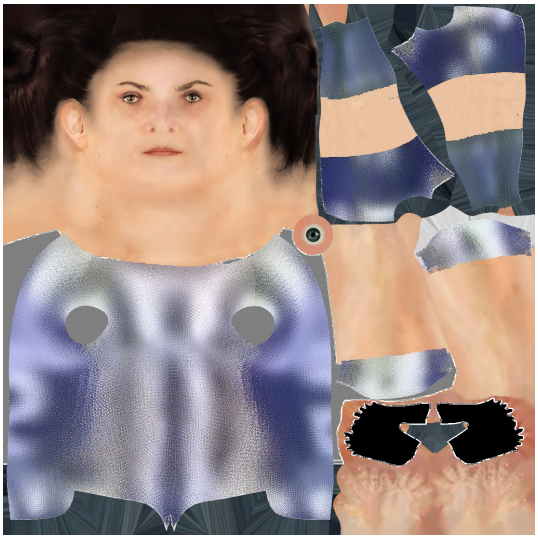
Fig. 48. Texture inpainting with SMPLitex on IFK player's partial texture.

The results show that the SMPLitex model correctly uses colours from relevant parts of the partial texture. It also produces wrinkles in the inpainted regions, perhaps best illustrated on the back of the IFK player in Fig. 48c. For the goalkeeper, whose kit is completely orange, the results are satisfactory and surpass a simple parameterised output in its visual fidelity. The dark regions at the boundaries around inpainted areas may be removed by either slightly expanding the inpaint mask or by applying a median filter to the result. On the other hand, both textures lack a number on their backs. While producing the correct number is impossible, hallucinating one would make the resulting texture more characteristic of a football player. Furthermore, the inpainted regions on the IFK player are not fully consistent with the detail in the partial texture. In particular, there are no stripes on the back of the texture.

These shortcomings are likely caused in part by the lack of football players in the training data. While fine-tuning a model generally does not require as much data as during pre-training, the fact that only 250 images were used likely limits the model's ability to faithfully represent a diverse set of appearances.

Image-to-Image in Texture Space

In addition to inpainting, the SMPLitex model supports image-to-image generation. This could prove useful for refining generated textures, removing grain and increasing contrast in the process, eliminating the need for the more naïve discretisation step. To evaluate its usefulness, we generate textures with the existing pipeline and feed them through the SMPLitex model with varying denoising strengths and the prompt "*a texture map of a soccer player*." The results are shown in Fig. 49.



(a) Original texture.

(b) Denoise: 0.3.

(c) Denoise: 0.4.

(d) Denoise: 0.5.

Fig. 49. Image-to-image generation with SMPLitex using 20 steps with a guidance scale of 14, with various amounts of noise added.

While the model does alter areas outside any clothing, these modifications will not affect the final texture but will be overwritten with the starting texture using the inverted clothing mask. To increase the saturation and the contrast between colours, the denoising strength must be made quite large. As a consequence, details are hallucinated and imperfections in the original texture such as slightly wavy stripes are exacerbated as opposed to toned down.

Guiding Texture Updates with Score Distillation Sampling (SDS)

Finally, we investigate the feasibility of implementing the SDS approach to refine textures generated with ℓ_1 loss. The primary goal is to reduce the patchy nature of details such as stripes on the player jerseys or the numbers on their backs, and for the textures to look more realistic. To this end, the NeRF model from the original formulation is replaced with the differentiable rendering pipeline, meaning the parameters ϕ now represent the texels of the texture to be optimised. Instead of using Imagen as the pre-trained diffusion model, we use the openly available Stable Diffusion 1.5 (SD 1.5) [33]. An inherent discrepancy is thus our use of a latent diffusion model. This means the noise will not be added to the image directly but to its latent embedding. A direct consequence of this is that the generated gradients must be backpropagated through the image encoder.

We initially evaluate our method qualitatively in two simpler scenarios. The first is to optimise the latents produced by the SD 1.5 image encoder given rendered images of players with the preliminary textures applied. The second is to optimise the pixels of those rendered images directly, requiring the propagation of gradients through the image encoder. We run the original pipeline for $2\,000$ iterations on one of the IFK players to generate a preliminary texture to be refined. The texture is rendered onto a posed SMPL mesh with a generic football pitch background. We then perform $1\,000$ iterations of SDS optimisation, in four separate runs: two where the image latents are optimised (first scenario), and two others where the pixels of the image are optimised directly (second scenario). In both scenarios, we compare two different guidance scales. The parameters are optimised using SGD with a learning rate of 0.01 and no momentum. The prompt is "*a DSLR photo of a soccer player*" and remains the same for all players to avoid the manual process of tuning the prompt to each player. The results of these experiments are shown in Fig. 50.

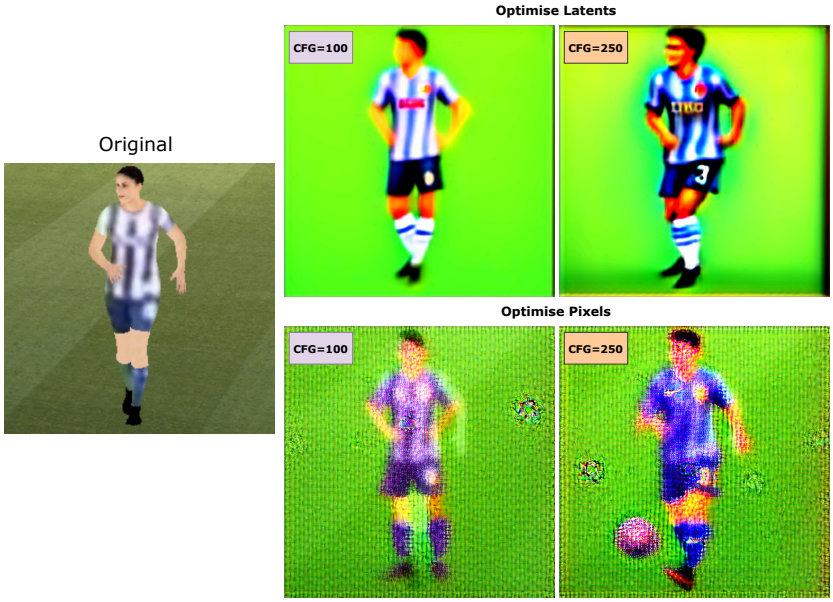


Fig. 50. Results from SDS experiments. Top row: optimised latents, decoded into images. Bottom row: input image pixels optimised directly. Both variants are attempted at two different guidance scales.

We observe high saturation in all cases, and more detailed results with higher guidance scales, both in line with observations by [43] and [53]. For example, in the case of optimising latents, the higher guidance scale of 250 results in more details in the face and the (hallucinated) number "3" on the shorts. Additionally, there is a striking difference in visual quality between the results from only optimising latents and optimising pixels. While the former generates smooth images, the latter produces high-frequency artefacts across the image. Considering the encoder in question is a convolutional neural network, such artefacts may be expected from backpropagation through its convolutional and downsampling layers [54]. Regardless of the guidance scale, the optimisation of image pixels does not enhance details such as stripes. Finally, none of the results are completely faithful to the original pose of the player.

To evaluate the utility of SDS loss for refining textures, the two phases of the texture generation pipeline are first run for an IFK player, with parameterised clothing followed by per-pixel optimisation with ℓ_1 and LPIPS loss. Then, two separate final passes of $1\,000$ iterations of SDS updates are run as in Fig. 51, with the same optimiser as the four previous tests. The first pass uses the same prompt of "*a DSLR photo of a soccer player*" as previously. The other pass uses a prompt more tuned to the player in view: "*a DSLR photo of a soccer player in blue and white striped jersey*." The resulting textures are applied to SMPL models and are shown in Fig. 52.

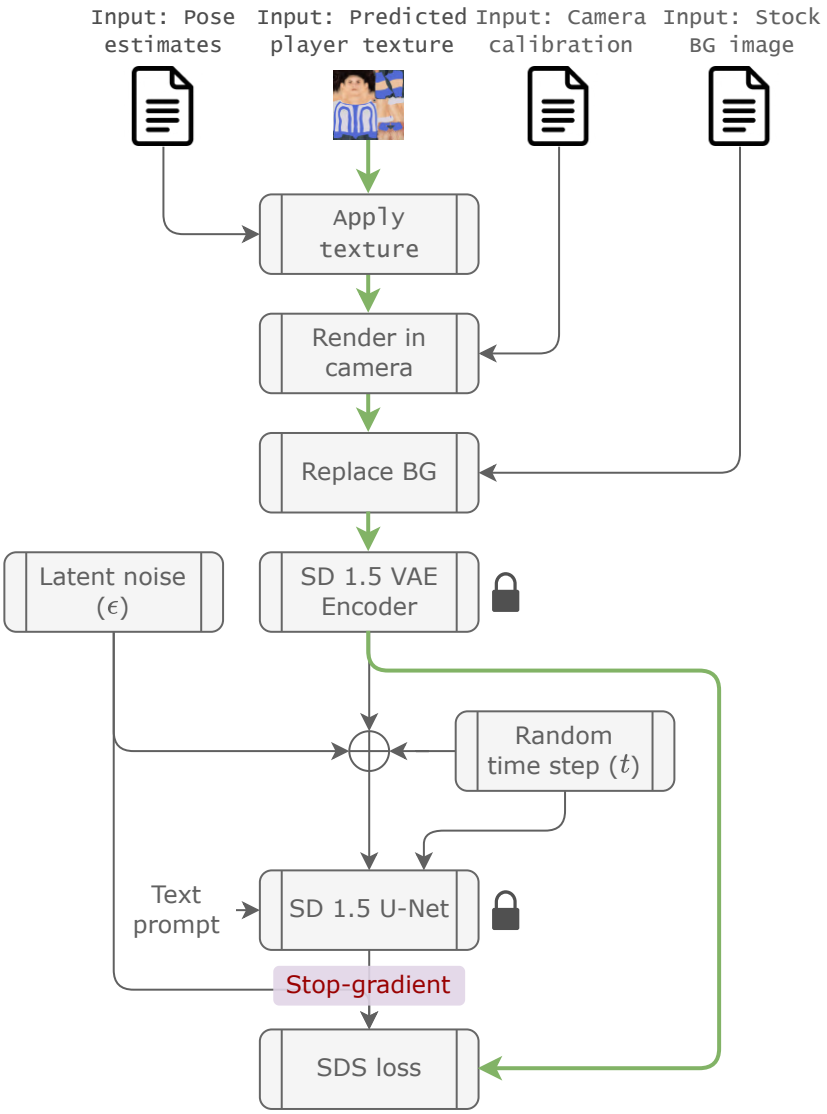
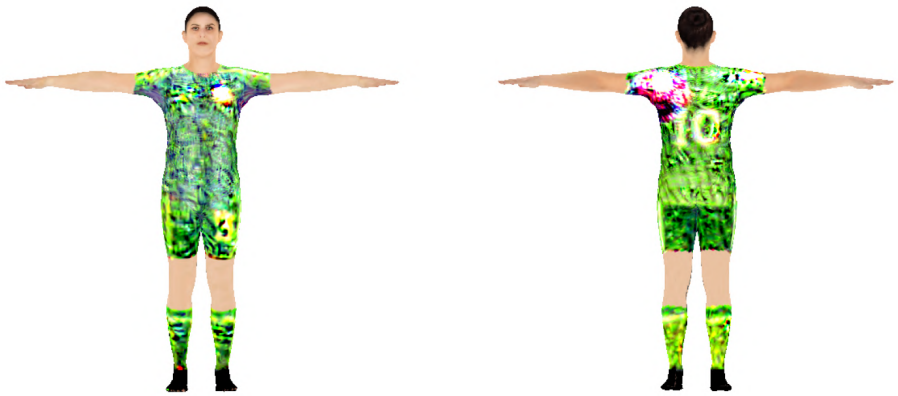
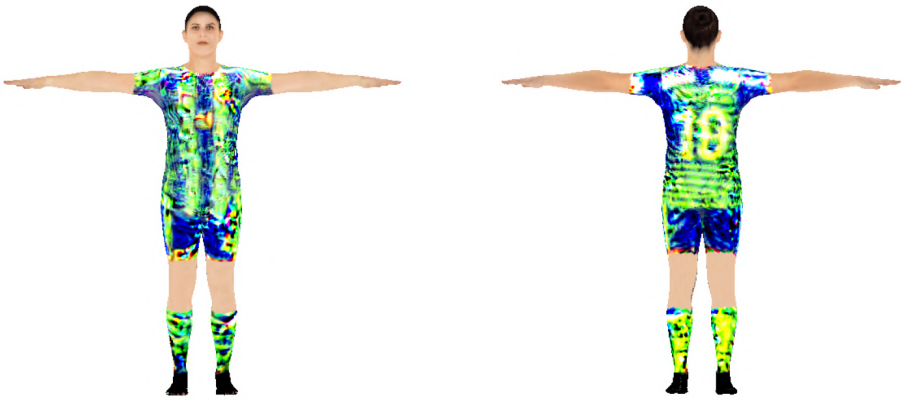


Fig. 51. SDS pipeline for texture refinement. The green path shows how gradients flow from the loss, through the image encoder, to the generated texture. No gradients flow through the denoising U-Net.



(a) Generic prompt.



(b) Tailored prompt.

Fig. 52. Textures after SDS passes with different prompts.

The texture does not improve after the SDS updates, likely as a consequence of the issues illustrated by the small-scale tests. Primarily, the tendency to alter the pose of the player in the input image may yield inconsistent results when evaluated from multiple viewing angles. The use of a more precise prompt did improve the results, but still results in incorrect colours and artefacts. Furthermore, a prompt that is tailored to each player is not a viable solution in a completely automated pipeline.

6

Conclusion

This thesis involved the construction of a virtual reality (VR) experience for the analysis of recorded football matches. It consists of a full-size football pitch with animated football players and a moving ball, with the pose estimation and ball tracking performed by concurrent work. The ball, which is tracked in the camera plane, is projected to the pitch utilising the camera parameters, which were also estimated by a concurrent work. The user can control the playback of the reconstruction, as well as position themselves at arbitrary locations on the pitch, including anchoring to the point-of-view of any player. By putting the user in the shoes of the football players, this immersive experience aims to bridge the gap between a video recording and real life.

Additionally, we have demonstrated the feasibility of extracting players' kits with differentiable rendering utilising the same pose estimates that are used for visualisation within the VR environment. This required overcoming challenges imposed by the low resolution of each player, slight deviations in alignment between the estimated pose and the player, as well as the limited number of viewing angles for each player, with many having parts hidden from the camera in all frames.

To address these, different loss functions were evaluated, including perceptual loss which improved the clarity of the generated textures. Using ℓ_1 loss also lead to a slight improvement in contrast over ℓ_2 loss. To further increase the contrast and saturation of the textures, colour palettes were extracted from the available images of each player, which, after a post-processing step, were used to limit the set of colours used within each texture. Parts of the textures that were not visible from the camera were inpainted using an off-the-shelf diffusion model fine-tuned on image textures of people. Finally, further refinement of textures using score distillation sampling was attempted, but requires further tuning to produce textures that are realistic and adhere to the attributes of the players in the images.

In conclusion, this thesis demonstrates significant advancements towards the objectives set out. The VR system successfully provides an immersive, interactive experience of football matches, allowing users to view and interact with the match from various angles and perspectives. While the reconstruction of high-resolution texture maps from low-quality images was only partially successful, the development of the method and the progress made highlight a promising direction for future work. By addressing the limitations and refining the techniques used, there is potential for achieving even more accurate and visually appealing textures in the future.

7

Future Research

The quality of the textures, particularly their value and saturation, would likely benefit from a more carefully designed loss function that better models the transformation of colours as they are captured in the camera.

To overcome the problem of parts of players not being visible, one may utilise the fact that the kits of players in the same team are nearly identical apart from the jersey numbers. All players of each team, apart from the goalkeeper, would then contribute to the same texture. Players whose backs were visible in at least one frame would be assigned personalised textures optimised from these views to reveal the number. The global orientation of the estimated pose may be used to determine whether a player's back is visible. The grouping of players into teams would likely require a type of clustering on their pixel colours similar to the current process for extracting colour palettes.

A future improvement to the application of denoising diffusion models is to add more effective guidance to the denoising process. This may be achieved by conditioning the denoiser on CLIP features generated from all patches belonging to any given player, instead of a generic text prompt. This sidesteps any requirement for text prompt engineering, while also tailoring the denoiser to visual features unique to each player. Additionally, using a joint loss function that combines SDS loss and, for example, LPIPS loss would likely have a similar effect. Finally, conditioning the diffusion models on additional attributes such as surface normals and silhouette, in the vein of previous works, may produce results that better adhere to the estimated pose. However, this would require training an auxiliary model like ControlNet.

References

[1] S. Fotbollförbundet. "Veo och svff i partnerskap – ökar streamingmöjligheterna i min fotboll." (2022), [Online]. Available: https://svff.svenskfotboll.se/nyheter/2022/11/veo-och-svff-i-partnerskap/.

[2] S. Fotbollförbundet. "Svff och analys- och livestreamingtjänsten spiideo i nytt samarbete." (2021), [Online]. Available: https://aktiva.svenskfotboll.se/nyheter/2021/08/SvFF-och-analys-och-livestreamingtjansten-Spiideo-i-nytt-samarbete/.

[3] M. Edlund, *Showcasing the vr demo*, Personal communication, 2024.

[4] H. G. Jhanzaib Humayun, "Tracking players and ball in football videos," MSc thesis, Department of Electrical Engineering, Chalmers University of Technology, Gothenburg, Sweden, 2024.

[5] O. S. Joakim Osterman, "3d pose estimation of football players," MSc thesis, Department of Electrical Engineering, Chalmers University of Technology, Gothenburg, Sweden, 2024.

[6] E. Arts, *Fifa 2024*, Video game, Available on various platforms including PlayStation, Xbox, and PC, 2023.

[7] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," in *Proceedings of the IEEE/CVF Conference on Computer Vision*, 2020.

[8] W. Jiang, K. M. Yi, G. Samei, O. Tuzel, and A. Ranjan, "Neuman: Neural human radiance field from a single video," in *European Conference on Computer Vision*, 2022.

[9] T. Jiang, X. Chen, J. Song, and O. Hilliges, "Instantavatar: Learning avatars from monocular video in 60 seconds," 2022.

[10] C. Guo, T. Jiang, X. Chen, J. Song, and O. Hilliges, "Vid2avatar: 3d avatar reconstruction from videos in the wild via self-supervised scene decomposition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

[11] A. Karthikeyan, R. Ren, Y. Kant, and I. Gilitschenski, "Avatarone: Monocular 3d human animation," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2024.

[12] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, "3d gaussian splatting for real-time radiance field rendering," *ACM Transactions on Graphics*, vol. 42, no. 4, Jul. 2023.

[13] M. Kocabas, J.-H. R. Chang, J. Gabriel, O. Tuzel, and A. Ranjan, "HUGS: Human gaussian splatting," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2024.

[14] J. Lei, Y. Wang, G. Pavlakos, L. Liu, and K. Daniilidis, "Gart: Gaussian articulated template models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

[15] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," vol. 33, 2020.

[16] I. Ho, J. Song, O. Hilliges, *et al.*, "Sith: Single-view textured human reconstruction with image-conditioned diffusion," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

[17] B. AlBahar, S. Saito, H.-Y. Tseng, C. Kim, J. Kopf, and J.-B. Huang, "Single-image 3d human digitization with shape-guided diffusion," in *Proceedings of the SIGGRAPH Asia 2023 Conference Papers*, 2023.

[18] Spiideo, *Sports video camera systems*, Available at `https://www.spiideo.com/sports-video-camera-systems/`, 2024.

[19] Veo, *Veo cam 3*, Available at `https://launch.veo.co/product/veo-cam-3`, 2024.

[20] J. Gomes, L. Velho, and M. C. Sousa, *Computer graphics: theory and practice*. Florida, USA: CRC Press, 2012.

[21] K. Lehn, M. Gotzes, and F. Klawonn, "Geometry processing," in *Introduction to Computer Graphics: Using OpenGL and Java*. Berlin, Germany: Springer Nature, 2023, pp. 129–192.

[22] H. Kato *et al.*, "Differentiable rendering: A survey," *ArXiv*, 2020.

[23] M. M. Loper and M. J. Black, "Opendr: An approximate differentiable renderer," in *Proceedings of the IEEE/CVF European Conference on Computer Vision*, 2014.

[24] S. Liu, T. Li, W. Chen, and H. Li, "Soft rasterizer: A differentiable renderer for image-based 3d reasoning," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019.

[25] N. Ravi *et al.*, "Accelerating 3d deep learning with pytorch3d," *arXiv*, 2020.

[26] J. Ansel *et al.*, "Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2024.

[27] *Blender*, Software, Version 4.0. Available at `https://www.blender.org/`, Blender Foundation, 2024.

[28] U. Technologies, *Unity game engine*, Software, Version 2023.2.17. Available at `https://unity.com/`, 2024.

[29] M. Loper, N. Mahmood, J. Romero, G. Pons-Moll, and M. J. Black, "SMPL: A skinned multi-person linear model," *ACM Trans. Graphics (Proc. SIGGRAPH Asia)*, vol. 34, no. 6, 248:1–248:16, Oct. 2015.

[30] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

[31] P. Dhariwal and A. Nichol, "Diffusion models beat gans on image synthesis," *Proceedings of the Conference on Advances in neural information processing systems*, 2021.

[32] J. Ho and T. Salimans, "Classifier-free diffusion guidance," in *Proceedings of the Conference on Neural Information Processing Systems*, 2021.

[33] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022.

[34] A. Vaswani *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[35] C. Saharia *et al.*, "Palette: Image-to-image diffusion models," in *Proceedings of the Conference on ACM SIGGRAPH 2022*, 2022, pp. 1–10.

[36] A. Radford *et al.*, "Learning transferable visual models from natural language supervision," in *Proceedings of the International Conference on Machine Learning*, 2021.

[37] L. Zhang, A. Rao, and M. Agrawala, "Adding conditional control to text-to-image diffusion models," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023.

[38] A. Kirillov *et al.*, "Segment anything," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023.

[39] Z. Cai *et al.*, "Smpler-x: Scaling up expressive human pose and shape estimation," in *Advances in Neural Information Processing Systems*, 2023.

[40] D. Casas and M. Comino-Trinidad, "SMPLitex: A Generative Model and Dataset for 3D Human Texture Estimation from Single Image," in *British Machine Vision Conference (BMVC)*, 2023.

[41] I. K. Rıza Alp Güler Natalia Neverova, "Densepose: Dense human pose estimation in the wild," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2018.

[42] X. Chen *et al.*, "Robust human matting via semantic guidance," in *Proceedings of the Asian Conference on Computer Vision (ACCV)*, 2022.

[43] B. Poole, A. Jain, J. T. Barron, and B. Mildenhall, "Dreamfusion: Text-to-3d using 2d diffusion," *arXiv*, 2022.

[44] C. Saharia *et al.*, "Photorealistic text-to-image diffusion models with deep language understanding," in *Advances in Neural Information Processing Systems*, 2022.

[45] G. Pavlakos *et al.*, "Expressive body capture: 3D hands, face, and body from a single image," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019.

[46] I. Meta Platforms, *Meta quest 3*, Virtual Reality Headset, Available at `https://www.meta.com/quest/quest-3/`, 2023.

[47] G. Bradski, "The opencv library.," *Dr. Dobb's Journal: Software Tools for the Professional Programmer*, vol. 25, no. 11, pp. 120–123, 2000.

[48] B. T. Phong, "Illumination for computer generated pictures," in *Seminal Graphics: Pioneering Efforts That Shaped the Field, Volume 1*. New York, NY, USA: Association for Computing Machinery, 1998, pp. 95–101.

[49] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The unreasonable effectiveness of deep features as a perceptual metric," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[50] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004. DOI: `10.1109/TIP.2003.819861`.

[51] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012.

[52] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv*, 2014.

[53] B. Tang, J. Wang, Z. Wu, and L. Zhang, "Stable score distillation for high-quality 3d generation," *arXiv*, 2023.

[54] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY