Dissertation Type: Research

DEPARTMENT OF COMPUTER SCIENCE

ARt-CG:

Assisted Real-time Content Generation of 3D Hair by Learning Manifolds

Dillon Keith Diep [INCOMPLETE DRAFT, NOT FOR SUBMISSION]

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Master of Engineering in the Faculty of Engineering.

Thursday 27th April, 2017

Declaration

This dissertation is submitted to the University of Bristol in accordance with the requirements of the degree of MEng in the Faculty of Engineering. It has not been submitted for any other degree or diploma of any examining body. Except where specifically acknowledged, it is all the work of the Author.

Dillon Keith Diep [INCOMPLETE DRAFT, NOT FOR SUBMISSION], Thursday 27$^{\text{th}}$ April, 2017

Contents

1 Contextual Background **1**
- 1.1 Production of 3D Content . . . 1
- 1.2 Curse of Dimensionality . . . 2
- 1.3 Related Research of Machine Learning in Creative Fields . . . 2
- 1.4 Motivation and Significance . . . 3
- 1.5 Challenges . . . 4
- 1.6 Central Objectives . . . 4

2 Technical Background **5**
- 2.1 Principal Component Analysis . . . 5
- 2.2 Probabilistic Principal Component Analysis . . . 5
- 2.3 Gaussian Process Latent Variable Model . . . 8
- 2.4 Bayesian Gaussian Process Latent Variable Model . . . 10
- 2.5 Formal Definition of 3D Polygon Mesh Representation . . . 11

3 Project Execution **13**
- 3.1 Training Data Set . . . 13
- 3.2 Generative Model of Hair . . . 14
- 3.3 Approximating Generative Parameters from Input Data . . . 15
- 3.4 Learning a Manifold with Bayesian GP-LVM . . . 19
- 3.5 Generation of Output . . . 19
- 3.6 Logging User Activity . . . 20
- 3.7 Project Management . . . 20

4 Critical Evaluation **21**
- 4.1 Functional Testing . . . 21
- 4.2 Behavioural Testing . . . 21
- 4.3 Statistical Analysis . . . 21
- 4.4 Evaluation . . . 21
- 4.5 Applications . . . 22

5 Conclusion **23**
- 5.1 Summary . . . 23
- 5.2 Project Status . . . 23
- 5.3 Future Work . . . 23

A An Example Appendix **27**

List of Figures

1.1 Image of hair geometry. 1

3.1 example polygon hair mesh training data. 13
3.2 (a) the original training mesh, (b) a partially retopologised mesh after tri-to-quad conversion operation, (c) a fully retopologised mesh after manually completing the conversion. . 14
3.3 3D spherical coordinate system 14
3.4 Sphere meshes visualise root positions. The angle ranges specify coverage area of the scalp, while the interval of placements determines resolution. Fitting a sphere to the reference head mesh approximates the radial distance and origin. 15
3.5 A possible configuration of 10260 parameters in the generative model, rendered on Plotly. 15
3.6 Our edge loop extraction algorithm begins by selecting an edge within the mesh graph. The two vertices of the edge are **end vertices**. We then proceed to *grow* the edge loop selection. Take the set of **first degree neighbours** of the *end vertices*, these nodes are the *candidates* for the edge loop. We remove edges that are part of the current edge loop from this set of first degree neighbours. We add the first degree neighbours to a set of face vertices for following iterations. Take the neighbours of the first degree neighbours as a set of **second degree neighbours**, removing its originating end vertex. A candidate node is only accepted to the edge loop if its set neighbours do not intersect with the set of face vertices. Accepted vertices are appended to the list of end vertices, this process is repeated until there are no end vertices left to grow the selection. 16
3.7 Observe that the corner vertices have two edges, and the boundary edges have three. Determining the root edge loop is done by choosing the boundary edge loop that is closest to the scalp surface. 17
3.8 Comparison of selection operators 18
3.9 GPy offers various standard kernels. Multi-modal kernels can be formed from combining kernels to represent data of complex nature. 19
3.10 As expected, a linearly embedded hairstyle is of little use as it can only extrapolate along one style. 19
3.11 Blender Latent Hair Modelling Add-on 20

List of Tables

3.1 This is an example table. 20

List of Algorithms

3.1 Parsing OBJ format 15
3.2 Extracting boundary edge loops 17
3.3 Determining root border edge loop 17
3.4 Extracting spline edge loops 17
3.5 Reconstructing spline path 17

List of Listings

Executive Summary

A compulsory section, of at most 1 **page**

The topic of this dissertation explores assisted content generation by training generative models using unsupervised machine learning for the production of 3D hair geometry. The research hypothesis of this study is that non-linear probabilistic principal component analysis with the Gaussian Process Latent Variable Model is applicable for improving the creative production workflow of complex 3D hair geometry for humanoids.

Production of 3D virtual worlds is a time-consuming and costly process that also demand expert knowledge. 3D assets encompass a vast range of applications, ranging from simulations and research, to contributing towards the functioning of many businesses. The production of 3D assets impacts many industries including engineering, medicine, and the provisioning of entertainment. One particular task is the creation of 3D hair geometry for humanoid characters. Creating 3D hair is arduous as hair structure is a complex system containing much interdependence between components.

Machine learning applications typically use large data sets for training on problems that often have a concise answer for a given prediction. The application of machine learning to enhance production for creative work is an exciting field that tackles novel challenges: artistic products tend to have small sets of data available and evaluation of quality is subjective. Given the same input, acceptable solutions can vary significantly. The outlined peculiarities of applying machine learning for 3D mesh data establish a unique field of problems to investigate.

Existing tools for 3D modelling have remained mostly static in the paradigm of approach over the past several decades. Automation through methods such as procedural generation can produce output much faster, but the lack of control over the final result makes it less desirable than traditional methods of 3D modelling. The focus of this project is to formulate a revolutionary approach that offers an alternative work-flow which could potentially improve the efficacy of producing 3D hair geometry through unsupervised training of generative models.

Deliverables:

- Formulation of a generative model for 3D humanoid hair structure.
- Resolving the alignment problem by approximating raw input data using a generative model.
- Learning a low dimension latent-space of high dimensional hair structure data.
- Investigated the performance of various kernels for high dimensional training data.
- Implemented an add-on package for a 3D production program, Blender.
  - The implementation creates guiding splines that are useful for generating hair geometry.
  - Appropriate for small training set that is practical for content creators.
  - Real-time performance that matches state of the art non-learning tools.
  - Functionality to records user activity for analysis.
- Acquired data for evaluation of the research hypothesis.

Supporting Technologies

A compulsory section, of at most 1 page

This section should present a detailed summary, in bullet point form, of any third-party resources (e.g., hardware and software components) used during the project. Use of such resources is always perfectly acceptable: the goal of this section is simply to be clear about how and where they are used, so that a clear assessment of your work can result. The content can focus on the project topic itself (rather, for example, than including "I used LaTeX to prepare my dissertation"); an example is as follows:

- I used the Java `BigInteger` class to support my implementation of RSA.
- I used a parts of the OpenCV computer vision library to capture images from a camera, and for various standard operations (e.g., threshold, edge detection).
- I used an FPGA device supplied by the Department, and altered it to support an open-source UART core obtained from http://opencores.org/.
- The web-interface component of my system was implemented by extending the open-source WordPress software available from http://wordpress.org/.

Notation and Acronyms

An optional section, of roughly 1 or 2 pages

Any well written document will introduce notation and acronyms before their use, *even if* they are standard in some way: this ensures any reader can understand the resulting self-contained content.

Said introduction can exist within the dissertation itself, wherever that is appropriate. For an acronym, this is typically achieved at the first point of use via "Advanced Encryption Standard (AES)" or similar, noting the capitalisation of relevant letters. However, it can be useful to include an additional, dedicated list at the start of the dissertation; the advantage of doing so is that you cannot mistakenly use an acronym before defining it. A limited example is as follows:

AES	:	Advanced Encryption Standard
DES	:	Data Encryption Standard
	⋮	
$\mathcal{H}(x)$	:	the Hamming weight of x
$\mathbb{F}_q$	:	a finite field with q elements
x_i	:	the i-th bit of some binary sequence x, st. $x_i \in \{0, 1\}$

x scalar
$\boldsymbol{x}$ column vector
$\boldsymbol{x}^T$ row vector, superscript T denotes transpose
$\boldsymbol{X}$ matrix
$\boldsymbol{I}$ identity matrix

Acknowledgements

An optional section, of at most 1 **page**

It is common practice (although totally optional) to acknowledge any third-party advice, contribution or influence you have found useful during your work. Examples include support from friends or family, the input of your Supervisor and/or Advisor, external organisations or persons who have supplied resources of some kind (e.g., funding, advice or time), and so on.

Chapter 1

Contextual Background

1.1 Production of 3D Content

There are many representations of 3D objects in computer graphics. A point cloud representation is a collection of points (vertices) that describe surface geometry. Range images map pixels of a depth image to a set of points in the scene. Voxels are unit cubes, corresponding to the concept of pixels, a collection of voxels describe an object volumetrically. These representations are common when sampling raw geometric data using 3D scanning devices. Each representation have advantages and disadvantages depending on the use case. It is possible to convert between representations, however, data loss may be incurred. The most precise representations are generally mathematical models such as NURBS (Non-uniform rational basis spline) - often applied in the CAD (Computer-Aided Design) industry. The most widely applied representation used for CGI (computer-generated imagery) are polygonal meshes.

In a production environment, it is straightforward to define geometry precisely to specified requirements as opposed to capturing examples. A polygon mesh is constructed of vertices, edges, and faces. The topology of a mesh concerns with the arrangement of its components, well-organised topology is required to maintain geometric qualities when processing the mesh with algorithmic operations. Polygon meshes are simple to define, yet combined with established techniques such as UV texture and normal mapping, are sufficiently expressive for visual purposes. Significant study has been conducted on the field of polygon meshes improve its versatility. In practice, polygon meshes are organised with quad-face topology (faces formed from four edges) during production. Quads conform better with editing tools and algorithms. The rendering pipeline often automatically converts polygon meshes to tri-faces (faces formed from three edges) as an optimisation process.

1.1.1 3D Hair Geometry

On average, a human is born with between 90,000 to 150,000 scalp hair follicles. [8] It is computationally very expensive to render and animate physically correct hair, but creative liberties have been taken to approximate, or stylise 3D hair such that it is both acceptable aesthetically and feasible in terms of performance. This study considers modelling of hair geometry, the motion of hair is assumed to be its default resting pose.

In recent years, impressive 3D hair solutions for real-time simulation of realistic hair and fur, such as *NVIDIA HairWorks* and *AMD TressFX* has emerged. These solutions, however, have limited application in comparison to their traditional counterpart of polygonal hair. It is often the case that texture-mapped polygonal hair is used as a fallback when advanced simulation fails. Realism is not necessarily always desirable, polygon hair can flexibly represent different art styles. In some cases, a blend of multiple representations are used to balance between cost and quality. 3D hair in cinematography with large budget can afford to render hair with much higher fidelity for important characters, but would still consider use efficient variants for scenarios such as crowd simulation. Ultimately, it can be observed that the representation of virtual hair generally follows a structure of splines with control points that define the overall organisation of strands or segments.

To add

Figure 1.1: Image of hair geometry.

1.2 Curse of Dimensionality

The curse of dimensionality is a term coined by Bellman, [**?**] which refers to the phenomena where exhibited complexity of high-dimensional spaces grows exponentially greater than those of low-dimensionality. From the perspective of a 3D content creator, manipulating high-dimensional data such as hair geometry is a daunting task that becomes increasingly difficult as more detail is added. Computationally, algorithms scale far better with low-dimensional data, causing complex geometry such as hair to be challenging for machine learning methods.

1.2.1 Procedural Generation and Automated Production

Procedural generation techniques produce output that adhere to rules established by the generative model defined. Such techniques have been successfully applied for generation of terrains and city modelling. [7] Fractals and methods such as the Lindenmayer system has been used to produce patterns that resemble those observed in nature. [**?**] Automated techniques such as the ones discussed, however, are seldom used for modelling important objects with specific design. It is difficult to control the output of procedurally generated content without heavily restricting its capabilities. Automated methods that do not learn is cannot adapt to changing demands without reimplementation.

1.3 Related Research of Machine Learning in Creative Fields

The task of machine learning can be divided into three major paradigms:

- *Supervised learning* is provided input training examples with desired outputs to learn the mapping of inputs to output.
- *Unsupervised learning* is given only input data, the procedure learns structure of and relation between data.
- *Reinforcement learning* seeks to iteratively improve a pool of solutions by simulating an environment that apply concepts inspired by the theory of evolution.

The role that learning methods play in both manufacturing and consumer application continue to grow, however, adoption has been slow for creative fields. Generally, robust models improve in performance as more reliable data is obtained. Creative production values uniqueness and versatility, properties that cause difficulty in machine learning methods. Varying artistic styles in design complicate feature analysis and ambiguity of correctness is problematic when predicting an output.

To overcome the challenges of the scenario introduced above, dimensionality reduction through unsupervised learning with probabilistic latent variable models such as the Gaussian Process Latent Variable Model (GP-LVM) [13] present an opportunity to learn stylistic properties of design and predict multiple acceptable outputs by analysing the likelihood. The following section will discuss related research conducted that also explores the idea of applying machine learning in creative fields.

1.3.1 Style-Based Inverse Kinematics

Style-based inverse kinematics introduced the Scaled Gaussian Process Latent Variable Model, based on GPLVM, to learn the probabilistic distribution of a 3D human posture model. [9] Character posing from motion data is represented as a 42-dimensional feature vector that encapsulated joint information of a humanoid body. Learning a model of poses established the relation between joints and identified constraints exhibited in the training data - where unusual postures are given a lower likelihood rating.

1.3.2 Latent Doodle Space

A latent doodle space is the use of a low-dimension latent space that has been applied on simple line drawings. [3] The motivation of a latent doodle space is to generating new drawings that are inspired by the input data. There are two key phases to derive a latent doodle space: the first challenge is to identify line strokes within drawings, a latent variable method is then used to learn a latent space.

1.3.3 Learning a Manifold of Fonts

A study by Campbell & Kautz presented a framework that learns the latent manifold of existing font styles. [5] The process involved universal parametrization of fonts to a polyline representation so that a distance measure is applicable and the generative model can interpolate between styles. Unsupervised learning with the GP-LVM model enabled rapid prototyping and non-experts could create font styles without experience on type design.

1.3.4 Real-time Drawing Assistance through Crowd-sourcing

Drawing assistance powered by large-scale crowd-sourcing explored the potential of data driven drawing to prompt for correction by achieving an artistic consensus. [14] A consensus is found by learning a correction vector field from training drawings. Stroke-correction is applied using the correction vector field to adjust user input dynamically.

1.3.5 AutoHair

Chai, et al introduced AutoHair, a method for automatic modelling of 3D hair from a portrait image. [6] The approach extracts information from images and uses a database of hair meshes to construct a 3D representation of the information conveyed. A hierarchical deep neural network trained on annotated hair images learn to segment hair and estimate growth direction within portraits. Data-driven hair matching and modelling algorithm fit meshes from the database to parameters output by the neural net model to automatically produce 3D hair. The experiment developed a traversable hairstyle space of 50,000 hair models, using training images and 3D exemplars obtained from the internet.

1.4 Motivation and Significance

State of the art 3D production software such as AutoDesk Maya, 3DS Max, and Blender are advanced programs with sophisticated list of features. That said, such programs have extremely convoluted user interfaces, even the most experienced professionals do not recognise each and every tool available. The most versatile tools are generally the most basic that perform atomic changes as they are applicable in every scenario. Examples include selection of primitives such as vertices, edges, or faces and performing translation, rotation, and scaling. Sculpting tools move many data points simultaneously, they are often used for defining organic surfaces now that modern machines are sufficiently powerful. Experienced artists might search for an existing base mesh that is similar to start on, but it is not always the case that such a base mesh exists - there are also concerns for quality, such as poor topology. As the geometry becomes more detailed and well-defined, each alteration makes less impact and the space of sensible changes becomes smaller. The design and production of 3D geometry remains a slow and delicate process.

Virtual hair creation is a necessity for characters of CG movies and video games that are embedded within culture both economically and as entertainment. Specialised artists learn to be proficient with the design of hair, variety of styles, and techniques for creating them. Hair geometry is much more concentrated than other types, containing many data points that is exhausting to edit. Soft selection and sculpting tools are good enough for defining the structure but maintaining topology and issues such as overlapping surfaces are still problematic. Learning the relation of hair structure allows the potential of discovering new hairstyles. It can also be used as a mean of rapidly generating initial base geometry that fits the target output better than existing geometry available. Generative methods could ensure a level of quality, clean topology that fits established specifications. Assisted content generation using machine learning provides a convenient, non-intrusive and intuitive method for rapidly generating new hair geometry from existing data.

The application of machine-learning based tools could enhance the workflow of professional users and improve the experience for non-expert consumers. Such tools integrate into the production environment to improve the efficiency of acquiring initial base geometry and visually compare designs during pre-production. Non-expert users receive the ability to produce 3D geometry without requiring to learn the intrinsics of traditional 3D modelling software. The rise of augmented reality and 3D printing inspires the development of generative tools that are intuitive and simplistic to use. Applications that allow users to create personal content could also integrate machine-learning based tools to prevent inappropriate or undesirable creation from being produced while providing options that surpass existing alternatives. An example would be avatar creation for many applications and video games. A space of reasonable options

generated from predefined outputs by the developers will allow users to interpolate between sensible configurations, providing an excellent level of customisation while adhering to defined constraints.

1.5 Challenges

This study faces a number of challenges. Firstly, 3D meshes are difficult to compare. The training data in its raw form will have varying dimensions. Meshes can be viewed as samples of the true geometry, thus meshes that represent the same object could differ drastically in number of data points depending on its level of detail. Typical feature extraction methods do not work well on meshes as artistic products are sensitive to data loss - any change could affect the perception of final result drastically.

Another problem encountered is the lack of training data. Typical machine learning solutions use huge data sets in the order of hundreds of thousands for training, but for 3D meshes the expected size of readily available training data is much smaller. Public repositories of 3D polygonal hair are generally around a few thousand in size. [11] Studios that store and organise past production could likely match the size of public repositories, depending on the size of the company. Private repositories of independent artists will rarely exceed the order of hundreds.

The application of machine learning methods must also account for subjectivity of evaluating artistic assets. The range of acceptable solutions is ambiguous, likened to how hair styles of characters can change drastically during the design phase, determining the threshold of acceptable solutions will be in itself a challenge to resolve.

As mentioned previously, 3D meshes are delicate and can easily be invalidated from small changes. Thus, reparations to ensure that the output of trained models are acceptable is a topic to explore.

In a production environment, the time required for a technique to return observable result directly affects throughput. For practical efficacy of assisted content generation, the technique should be reasonably fast in presenting observable output.

1.6 Central Objectives

- Resolving the alignment problem of 3D data through a representative generative model.
- Explore the application of GP-LVM for 3D hair geometry in a production pipeline.
- Investigate the use of latent variables for identifying stylistic properties of 3D hair geometry.
- Demonstrate the use of non-linear manifold to generate new hairstyles from training data.
- Enable an intuitive method for non-experts to create 3D hair geometry.
- Observable output demand performance close to real-time for practical use.

Chapter 2

Technical Background

2.1 Principal Component Analysis

In multivariate analysis, principal component analysis (PCA) is a statistical technique used to perform linear dimensionality reduction. It was originally introduced by Pearson [10], and independently developed by Hotelling [**?**], where the standard algebraic derivation of PCA was presented in terms of a standardized linear projection.

Consider the properties that describe hair structure. Observable variables that can be measured include location, orientation, length, and colour. The data collected may indicate that some variables change together, this relation is measured as the covariance. The PCA technique searches for an ordered set of principal components that retain maximal variance. A principal component can be viewed as a combination of the observed variables. Should two observed variables strongly covary linearly, then it is plausible to describe the data with a single variable instead. The more linearly observed variables covary, the less information is lost from choosing a smaller set of principal components, thus effectively reducing dimensionality.

Given a set of n observed d-dimensional data represented as a design matrix, $\boldsymbol{X} = [\boldsymbol{x_1}, ..., \boldsymbol{x_n}]^T$, the q principal components $\boldsymbol{w_j}$, $j \in 1, ..., q$, are the orthonormal axes with maximal variance retained. The first principal component is a linear function $\boldsymbol{\alpha}_1^T \boldsymbol{X}$ that retains most variance of $\boldsymbol{X}$, where $\boldsymbol{\alpha} = [\alpha_{11}, \alpha_{12}, ..., \alpha_{1n}]$ is a vector of n constants such that:

$$\boldsymbol{\alpha}_1^T \boldsymbol{X} = \alpha_{11}\boldsymbol{x_1} + \alpha_{12}\boldsymbol{x_2} + ... + \alpha_{1n}\boldsymbol{x_n} = \sum_{i=1}^{n} a_{1i}\boldsymbol{x_i}$$

The following principal components are found by looking for a linear function that is orthogonal to the selected principal components and retain maximum variance.

PCA can be performed by singular value decomposition (SVD) of $\boldsymbol{X}$ in matrix form, [12]

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{L}\boldsymbol{V}^T,$$

where given $r = r(\boldsymbol{X})$ denotes the rank of $\boldsymbol{X}$, then $\boldsymbol{U} \in \Re^{n \times r}$ is a matrix of orthonormal columns that are the left singular vectors, $\boldsymbol{L} \in \Re^{r \times r}$ is a diagonal matrix of the singular values of $\boldsymbol{X}$, and $\boldsymbol{V} \in \Re^{d \times r}$ is a matrix of orthonormal columns that are the right singular vectors.

2.2 Probabilistic Principal Component Analysis

A limitation of standard PCA is the lack of a probabilistic solution. Tipping and Bishop introduced a probabilistic principal component analysis by constraining the noise distribution of a latent variable model to be Gaussian such that it is effectively equivalent when its marginal likelihood is maximised. [18]

Suppose that the PCA is applied to a set of data that represents hair structure. Standard PCA searches for the orthonormal axes that retain maximal variance. Extrapolating values using the principal axes can infer new hairstyles, but only within a fixed style described by the principal axes. In the case of 3D content production, it is much more useful to have a selection of plausible designs to choose from. A probabilistic model of PCA will enable exploration of other potential linear embeddings of hair structure.

2.2.1 The Gaussian Distribution

The Gaussian (normal) distribution is a reasonable prior assumption for data that is subject to the central limit theorem, which states that as the sample size of a population tends to infinity, the distribution becomes increasingly Gaussian. [4, p.78] A random variable X that is normally distributed with mean μ and variance σ^2 is denoted as

$$X \sim \mathcal{N}(\mu, \sigma^2).$$

The Gaussian density for a single variable y is expressed as [4, p.78]:

$$\mathcal{N}(y|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y-\mu)^2}{2\sigma^2}\right),$$
$$\mathcal{N}(y|\mu, \sigma^2) \equiv p(y|\mu, \sigma^2).$$

Properties of Gaussian Distribution

Notable properties of the Gaussian distribution that are useful include the summation (2.1), scaling (2.2), and product (2.3) operation - all of which yields a result that is also a Gaussian distribution. [17, p.200]

$$\sum_{i=1}^{n} y_i \sim \mathcal{N}(\sum_{i=1}^{n} \mu_i, \sum_{i=1}^{n} \sigma_i^2) \tag{2.1}$$

$$wy \sim \mathcal{N}(w\mu, w^2\sigma^2) \tag{2.2}$$

$$\mathcal{N}(\boldsymbol{x}|\boldsymbol{a}, A)\mathcal{N}(\boldsymbol{x}|\boldsymbol{b}, B) = \mathcal{N}(\boldsymbol{a}|\boldsymbol{b}, A+B)\mathcal{N}(\boldsymbol{x}|\boldsymbol{c}, C), \tag{2.3}$$
$$\boldsymbol{c} = C(A^{-1}\boldsymbol{a} + B^{-1}\boldsymbol{b}),$$
$$C = (A^{-1} + B^{-1})^{-1}).$$

Multivariate Gaussian Distribution

Let w and h be jointly Gaussian distributed variables, if the variables are independent, then $p(w,h) = p(w)p(h)$. The joint probability density is thus,

$$p(w,h) = \frac{1}{\sqrt{2\pi\sigma_1^2}\sqrt{2\pi\sigma_2^2}} \exp\left(-\frac{1}{2}\left(\frac{w-\mu_1)^2}{\sigma_1^2} + \frac{h-\mu_2)^2}{\sigma_2^2}\right)\right).$$

In matrix form, the joint probability is

$$p(w,h) = \frac{1}{2\pi\sqrt{\sigma_1^2\sigma_2^2}} \exp\left(-\frac{1}{2}\left(\begin{bmatrix} w \\ h \end{bmatrix} - \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}\right)^T \begin{bmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{bmatrix} \left(\begin{bmatrix} w \\ h \end{bmatrix} - \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}\right)\right).$$

For a n-dimensional vector $\boldsymbol{y}$, the joint probability density is expressed as

$$p(\boldsymbol{y}) = \frac{1}{2\pi|\boldsymbol{D}|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{y}-\boldsymbol{\mu})^T\boldsymbol{D}^{-1}(\boldsymbol{y}-\boldsymbol{\mu})\right), \tag{2.4}$$

where $\boldsymbol{D} \in \Re^{n\times 1}$ is the diagonal matrix of the variances Σ. [4, p.78]

2.2.2 Latent Variable Models and Factor Analysis

A latent variable model transforms a set of n d-dimensional observed variables encoded as a design matrix, $\boldsymbol{Y} \in \Re^{n\times d}$, to a set of n q-dimensional latent (unobserved) variables, $\boldsymbol{X} \in \Re^{n\times q}$. Latent variables are parsimonious, it is generally the case that $q \ll d$, explaining the original data with fewer variables. A notable latent variable model is that of factor analysis, one that assumes linearity in relation of the observed data set,

$$\boldsymbol{Y} = \boldsymbol{W}\boldsymbol{X} + \boldsymbol{\mu} + \boldsymbol{\epsilon}. \tag{2.5}$$

$\boldsymbol{W}$ represents a matrix that specifies the linear relation between the observed data-space with the latent-space. The parameter μ allows for non-zero mean, and the ϵ parameter represents noise within the model. Standard PCA can be viewed as a variant of factor analysis where the noise parameter is not accounted for. The maximum-likelihood estimates of $\boldsymbol{W}$ will thus generally not correspond to the principal subspace.

2.2.3 A Probabilistic Model for PCA

Tipping and Bishop develop a latent variable model that performs principal component analysis by modelling the parameter ϵ of equation 2.5 as an isotropic, spherical Gaussian distribution $\mathcal{N}(\mathbf{0}, \beta^{-1}\boldsymbol{I})$. Consider a design matrix of n centred d-dimensional vectors $\boldsymbol{Y} = [\boldsymbol{y_1}, ..., \boldsymbol{y_n}]^T$. For each observed data point, $1 \leq i \leq n$, there is an associated q-dimensional latent variable $\boldsymbol{x_i}$. The original data can be represented in terms of the latent variable with noise value,

$$\boldsymbol{y}_i = \boldsymbol{W}\boldsymbol{x}_i + \boldsymbol{\epsilon}_i.$$

A general representation for data with non-zero mean includes the mean parameter, $\boldsymbol{\mu}_i$,

$$\boldsymbol{y}_i = \boldsymbol{W}\boldsymbol{x}_i + \boldsymbol{\mu}_i + \boldsymbol{\epsilon}_i.$$

The matrix $\boldsymbol{W} \in \boldsymbol{\Re^{d \times q}}$ represents the linear relationship between the latent-space with the data-space. The noise values, $\epsilon_i \in \Re^{d \times 1}$, are sampled from a independent spherical Gaussian distribution, $\epsilon_i \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\beta^{-1}I})$. The conditional probability distribution of a observed variables $\boldsymbol{y}_i$ given input variables $\boldsymbol{x}_i$ is thus Gaussian distributed as

$$p(\boldsymbol{y}_i|\boldsymbol{x}_i) = \mathcal{N}(\boldsymbol{W}\boldsymbol{x_i} + \boldsymbol{\mu_i}, \boldsymbol{\beta}^{-1}\boldsymbol{I}). \tag{2.6}$$

The marginal distribution over the latent variables are standard Gaussian, defined as $\boldsymbol{x}_i \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. The marginal distribution for the observed data $\boldsymbol{y}_i$ is obtained by integrating out the latent variables. From equation 2.4, an arbitrary rotation matrix $\boldsymbol{R}^T \in \Re^{n \times n}$ can be applied to the basis, forming the correlated Gaussian,

$$p(\boldsymbol{y}_i) = \frac{1}{2\pi|\boldsymbol{D}|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{R}^T\boldsymbol{y}_i - \boldsymbol{R}^T\boldsymbol{\mu}_i)^T\boldsymbol{D}^{-1}(\boldsymbol{R}^T\boldsymbol{y}_i - \boldsymbol{R}^T\boldsymbol{\mu}_i)\right).$$

This gives an eigenvalue decomposition of the inverse covariance matrix, and thus the covariance matrix,

$$\boldsymbol{C}^{-1} = \boldsymbol{R}\boldsymbol{D}^{-1}\boldsymbol{R}^T,$$

$$\boldsymbol{C} = \boldsymbol{R}\boldsymbol{D}\boldsymbol{R}^T.$$

As a consequence, we can derive that given $\boldsymbol{x}_i \sim \mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\beta}^{-1})$ and $\boldsymbol{y}_i = \boldsymbol{W}\boldsymbol{x}_i$, then the distribution of the observed variables can be denoted as $\boldsymbol{y}_i \sim \mathcal{N}(\boldsymbol{W}\boldsymbol{\mu}_i, \boldsymbol{W}\boldsymbol{\beta}^{-1}\boldsymbol{W}^T)$. Thus, for centred data where $\boldsymbol{\mu} = \mathbf{0}$,

$$\boldsymbol{W}\boldsymbol{x}_i \sim \mathcal{N}(\mathbf{0}, \boldsymbol{W}\boldsymbol{W}^T),$$

$$\boldsymbol{y}_i \sim \mathcal{N}(\mathbf{0}, \boldsymbol{C}),$$

where the observation covariance model is $\boldsymbol{C} = \boldsymbol{W}\boldsymbol{W}^T\beta^{-1}\boldsymbol{I}$, with corresponding log-likelihood [13]

$$\mathcal{L} = \frac{n}{2}(dln(2\pi) + ln|\boldsymbol{C}| + tr(\boldsymbol{C^{-1}S})), \tag{2.7}$$

$$\boldsymbol{S} = \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{y}_i - \mu)(\boldsymbol{y}_i - \mu)^T.$$

We can write the likelihood for a data point as

$$p(\boldsymbol{y_i}|\boldsymbol{x_i}, \boldsymbol{W}, \boldsymbol{\beta}) = \boldsymbol{\mathcal{N}(y_i}|\boldsymbol{W}x_i, \boldsymbol{\beta^{-1}I}). \tag{2.8}$$

Integrating over the latent variables gives the marginal likelihood,

$$p(\boldsymbol{y_i}|\boldsymbol{W}, \boldsymbol{\beta}) = \int p(\boldsymbol{y_i}|\boldsymbol{x_i}, \boldsymbol{W}, \boldsymbol{\beta})p(\boldsymbol{x_i})d\boldsymbol{x_i}.$$

As the prior of probabilistic PCA is modelled as a standard Gaussian distribution, $p(\boldsymbol{x_i}) = \mathcal{N}(\boldsymbol{x_i}|\mathbf{0}, \boldsymbol{I})$, marginalisation of the integral obtains the marginal likelihood of each data point as

$$p(\boldsymbol{y_i}|\boldsymbol{W}, \boldsymbol{\beta}) = \mathcal{N}(\boldsymbol{y_i}|\mathbf{0}, \boldsymbol{WW^T} + \boldsymbol{\beta^{-1}I}).$$

Assuming that the data points are independent, the likelihood of the full data set is the product of each marginal likelihood,

$$p(\boldsymbol{Y}|\boldsymbol{W}, \boldsymbol{\beta}) = \prod_{i=1}^{n} p(\boldsymbol{y_i}|\boldsymbol{W}, \boldsymbol{\beta}).$$

2.2.4 The Principal Subspace of PPCA

Tipping and Bishop[18] showed that all potential solutions for $\boldsymbol{W}$, the likelihood (2.7), is of the form

$$\boldsymbol{W} = \boldsymbol{U_q}(\boldsymbol{K_q} - \boldsymbol{\sigma^2 I})^{\frac{1}{2}}\boldsymbol{R}.$$

One particular case of interest is when the likelihood is maximised,

$$\boldsymbol{W}_{ML} = \boldsymbol{U}_q\boldsymbol{L}\boldsymbol{R}, \tag{2.9}$$

$$\boldsymbol{L} = (\Lambda_q - \sigma^2\boldsymbol{I})^{\frac{1}{2}}$$

The matrix $\boldsymbol{U_q} \in \Re^{d\times q}$ contains the column vectors that are the principal eigenvectors, $\Lambda_q = [\lambda_1, ..., \lambda_q]$ represents the diagonal matrix of the corresponding eigenvalues, and $\boldsymbol{R} \in \Re^{q\times q}$ represent an arbitrary orthogonal rotation matrix.

Maximising the likelihood of $\boldsymbol{W}$ by equation 2.9 on the latent variable model defined by equation 2.5 maps the latent-space to the principal subspace of the observed data. Selecting $\boldsymbol{W}_{ML}$, the latent variable model is effectively equivalent to standard principal component analysis.

2.3 Gaussian Process Latent Variable Model

The Gaussian Process Latent Variable Model (GP-LVM) is a non-linear latent variable model derived from a dual of the probabilistic PCA by replacing the inner product kernel with Gaussian processes (Lawrence 2005) [13]. The use of an inner product kernel that allows for non-linear functions enable a non-linear embedding of the data-space to obtain a lower dimension latent-space.

Probabilistic PCA presented a space of linearly embedded hairstyles, however, it is unlikely that linear embeddings of observed variables for hair is effective for high dimensional data such as hair structure. GP-LVM replaces the covariance matrix with a non-linear kernel, allowing non-linear embedding of the observed variables to capture the complexity of hair structure.

2.3.1 Dual Probabilistic PCA

The dual probabilistic PCA introduced by Lawrence marginalises the parameters, $\boldsymbol{W}$, and optimises with respect to latent variables, $\boldsymbol{X}$. This is the dual approach of the standard probabilistic PCA where the parameters are optimised and the latent variables are marginalised.

First, a conjugate prior to the likelihood of probabilistic PCA (2.8) is taken to be a spherical Gaussian distribution,

$$p(\boldsymbol{W}) = \prod_{i=1}^{d} \mathcal{N}(\boldsymbol{w}_i|\boldsymbol{0}, \boldsymbol{I}).$$

As marginalisation of both $\boldsymbol{W}$ and $\boldsymbol{X}$ is often intractable in practice, $\boldsymbol{W}$ is selected for marginalisation as the conjugate prior is Gaussian distributed, thus, it can be integrated analytically. The marginalised likelihood of $\boldsymbol{W}$ is

$$p(\boldsymbol{Y}|\boldsymbol{X}, \beta) = \prod_{i=1}^{d} p(\boldsymbol{y}_{:,i}|\boldsymbol{X}, \beta),$$

The $\boldsymbol{y}_{:,i}$ parameter represents the i^{it} column of $\boldsymbol{Y}$, where

$$p(\boldsymbol{y}_{:,i}|\boldsymbol{X}, \beta) = \mathcal{N}(\boldsymbol{y}_{:,i}|\boldsymbol{0}, \boldsymbol{X}\boldsymbol{X}^T + \beta^{-1}\boldsymbol{I}).$$

The objective function is the log-likelihood

$$L = -\frac{dn}{2}ln2\pi - \frac{d}{2}ln|\boldsymbol{K}| - \frac{1}{2}tr(\boldsymbol{K^{-1}}\boldsymbol{Y}\boldsymbol{Y}^T), \tag{2.10}$$

$$\boldsymbol{K} = \boldsymbol{X}\boldsymbol{X}^T + \beta^{-1}I.$$

In the original paper, Lawrence found the gradients of the log-likelihood (2.10) with respect to $\boldsymbol{X}$ as

$$\frac{\sigma L}{\sigma \boldsymbol{X}} = \boldsymbol{K^{-1}}\boldsymbol{Y}\boldsymbol{Y}^T\boldsymbol{K^{-1}}\boldsymbol{X} - d\boldsymbol{K^{-1}}\boldsymbol{X}.$$

A stationary point where the gradients are zero is given by

$$\frac{1}{d}\boldsymbol{Y}\boldsymbol{Y}^T\boldsymbol{K}^{-1}\boldsymbol{X} = \boldsymbol{X}.$$

The values for $\boldsymbol{X}$ which maximise the likelihood are given by singular value decomposition of $\boldsymbol{X}$,

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{L}\boldsymbol{V}^T.$$

$\boldsymbol{U}$ is an $n \times q$ matrix whose orthonormal column vectors are the first eigenvectors of $\boldsymbol{Y}\boldsymbol{Y}^T$. $\boldsymbol{L}$ is a $q \times q$ diagonal matrix of singular values, whose j^{th} element is $l_j = (\lambda_j - \frac{1}{\beta})^{-\frac{1}{2}}$, where λ_j is the eigenvalue associated with the j^{th} eigenvector $d^{-1}\boldsymbol{Y}\boldsymbol{Y}^T$. $\boldsymbol{V}$ is an arbitrary $q \times q$ rotation matrix. Lawrence showed that the eigenvalue problem developed here is equivalent to the eigenvalue problem solved in PPCA, and thus, DPPCA is also effectively equal to standard PCA when the likelihood is maximised.

Dual probabilistic PCA assumes that the output dimensions are linear, independent, and identically distributed. Infringing upon these assumptions derive new probabilistic models.

2.3.2 Gaussian Processes

A Gaussian process (GP) is a non-parametric statistical model that generalises a probability distribution over functions by treating each observed variable as an independent distribution. Formally, a Gaussian process is a finite collection of random variables that are jointly Gaussian, specified by a mean function $m(\boldsymbol{x})$ and covariance function $k(\boldsymbol{x}, \boldsymbol{x'})$ of a real process $f(\boldsymbol{x})$, [17, p.13]

$$f(\boldsymbol{x}) \sim \mathcal{GP}(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x'})),$$

where

$$m(\boldsymbol{x}) = \mathbb{E}[f(\boldsymbol{x})],$$

$$k(\boldsymbol{x}, \boldsymbol{x'}) = \mathbb{E}[(f(\boldsymbol{x}) - m(\boldsymbol{x}))(f(\boldsymbol{x'}) - m(\boldsymbol{x'}))].$$

The mean function of a Gaussian process is generally assumed to be zero, unless stated otherwise. Polynomial regression models yield best results when the behaviour of the observed data resembles the selected polynomial function, but determining a suitable function is challenging. As a non-parametric model, Gaussian processes provides a probability distribution over a space of functions that associates a likelihood for each function, presenting the opportunity to sample various functions that encode the behaviour of observed data. Gaussian process regression models can be overfitted, but overfitting a space of functions is more lenient than overfitting a single function.

GPs for Regression

The observed data $\boldsymbol{y}$ is assumed to be modelled by a function $f(\boldsymbol{x})$ with input $\boldsymbol{x}$ and corrupted by noise, ϵ. Noise interference depends on the problem, a simple relationship is a model with additive noise,

$$y = f(\boldsymbol{x}) + \epsilon.$$

Suppose that the noise on each observed variable is modelled by an independent Gaussian distribution, $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$, the joint likelihood is the product of marginal likelihoods,

$$p(y|X, w) = \prod_{i=1}^{n} p(y_i|x_i, w) = \mathcal{N}(X^T w, \sigma_n^2 \boldsymbol{I})$$

In Bayesian reasoning, we update our prior knowledge with observed evidence to obtain a posterior. This methodology is expressed in Bayes' Theorem,

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{marginal likelihood}}, \qquad p(w|y, X) = \frac{p(y|X, w)p(w)}{p(y|X)} \tag{2.11}$$

The marginal likelihood $p(y|X)$ is a normalising constant, given by integral

$$p(y|X) = \int p(y|X, w)p(w)dw.$$

The posterior expresses what we know about the parameters using the likelihood and the prior. We can then use the posterior to make an informed prediction for test inputs.

The dual probabilistic PCA model uses a Gaussian process prior that is corrupted by Gaussian noise, [13] $\epsilon \sim \mathcal{N}(\mathbf{0}, \beta^{-1}\boldsymbol{I})$. The covariance function (kernel) is thus,

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i^T \boldsymbol{x}_j + \beta^{-1}\delta_{ij}. \tag{2.12}$$

Parameters $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are vectors from the space of inputs to the function and $\sigma_1 ij$ represents the Knronecker delta, defined by

$$\delta_{ij} = \begin{cases} 1, & \text{if } i = j, \\ 0, & \text{if } i \neq j. \end{cases} \tag{2.13}$$

Taking inputs from matrix $\boldsymbol{X}$ and evaluating the covariance function at each observed variable gives the covariance matrix,

$$\boldsymbol{K} = \boldsymbol{X}\boldsymbol{X}^T + \beta^{-1}\boldsymbol{I}.$$

The element at i_{th} row and j_{th} column of $\boldsymbol{K}$ is given by the prior distribution (2.12). Thus, the marginal likelihood of dual probabilistic PCA is a product of d independent Gaussian processes.

Learning with the Covariance Function

The covariance function of a Gaussian process describes the properties of functions, such as variability. Learning in Gaussian processes is to determine a suitable hyperparameters of a covariance function.

2.4 Bayesian Gaussian Process Latent Variable Model

The Bayesian Gaussian Process Latent Variable Model (Bayesian GP-LVM) [19] extends the GP-LVM by variationally integrating out the input variables of the Gaussian process to approximate the marginal likelihood of a fully marginalised model. The approximated marginal likelihood can be used to compute a lower bound that is robust to overfitting. A fully marginalised model establishes a Bayesian perspective that copes well with uncertainty caused by missing data and can automatically determine latent dimensions within the observed data set.

The marginalised likelihood of the GP-LVM can be represented in the form

$$p(\boldsymbol{Y}|\boldsymbol{X}) = \prod_{i=1}^{d} p(\boldsymbol{y}_{:,i}|\boldsymbol{X})$$

where $\boldsymbol{y}_{:,i}$ represents the i^{th} column of $\boldsymbol{Y}$ and

$$p(\boldsymbol{y}_{:,i}|\boldsymbol{X}) = \mathcal{N}(\boldsymbol{y}_{:,\boldsymbol{i}}|\mathbf{0}, K_{nn} + \beta^{-1}\boldsymbol{I}_n).$$

K_{nn} is the $n \times n$ covariance matrix defined by the kernel function $k(\boldsymbol{x}, \boldsymbol{x'})$. The latent variable X is assigned a prior density given by the standard Gaussian distribution,

$$p(\boldsymbol{X}) = \prod_{i=1}^{n} \mathcal{N}(\boldsymbol{x_i}|\mathbf{0}, \boldsymbol{I}).$$

Each $\boldsymbol{x_i}$ is the i^{th} row of $\boldsymbol{X}$. The joint probability model for the GP-LVM is

$$p(\boldsymbol{Y}, \boldsymbol{X}) = p(\boldsymbol{Y}|\boldsymbol{X})p(\boldsymbol{X})$$

The standard GP-LVM method trains by finding the MAP estimate of $\boldsymbol{X}$ whilst jointly maximizing with respect to the hyperparameters. Bayesian GP-LVM performs variational inference to marginalise the latent variables. This method enables optimisation of the resulting lower bound on the marginal likelihood with respect to the hyperparameters.

2.4.1 Variational Inference

In order to apply variational Bayesian methods to GP-LVM, the latent/input variables that appear non-linearly must first be approximately integrated out.

The marginal likelihood of the observed data is obtained by integrating out the latent variables:

$$p(\boldsymbol{Y}) = \int p(\boldsymbol{Y}|\boldsymbol{X})p(\boldsymbol{X})d\boldsymbol{X}.$$

Computationally, this integration is intractable in practice. Variaitonal Bayesian methods can instead be used by using variational distribution $q(\boldsymbol{X})$ to approximate the posterior distribution over the latent variables, $p(\boldsymbol{X}|\boldsymbol{Y})$.

$$q(\boldsymbol{X}) = \prod_{i=1}^{n} \mathcal{N}(\boldsymbol{x_n}|\boldsymbol{\mu_n}, \boldsymbol{S_n}).$$

The variational parameters are $\{\boldsymbol{\mu_n}, \boldsymbol{S_n}\}_{\boldsymbol{i=1}}^{\boldsymbol{n}}$ and $\boldsymbol{S_n}$ is a diagonal covariance matrix.

The variational distribution can then be used to obtain a Jensen's lower bound on $\log p(\boldsymbol{Y})$:

$$F(q) = \int q(\boldsymbol{X}) \log \frac{p(\boldsymbol{Y}|\boldsymbol{X})p(\boldsymbol{X})}{q(\boldsymbol{X})} d\boldsymbol{X}$$

$$= \int q(\boldsymbol{X}) \log p(\boldsymbol{Y}|\boldsymbol{X})p(\boldsymbol{X})d\boldsymbol{X} - \int q(x) \log \frac{q(X)}{p(X)} dX$$

$$= \tilde{F}(q) - KL(q||p).$$

The $KL(q||p)$ term is the negative KL divergence between the variational posterior distribution $q(X)$ and the prior distribution p(X) over the latent variables. Since the distributions are Gaussian, the negative KL divergence is tractable. The problematic term is $\tilde{F}(q)$, where variational sparse Gaussian process regression is applied for approximation.

2.5 Formal Definition of 3D Polygon Mesh Representation

2.5.1 The Mesh Data Structure

Polygon mesh data structure used in contemporary 3D programs vary by implementation. The winged-edge polyhedron representation defined by Baumgart in 1972 [2] specified the essential components required to model a polygonal mesh: vertices, edges, and faces. We take advantage of the structural similarity between a mesh and a graph to inspire a formal definition for polygonal mesh geometry.

Definition of a Graph

A **graph** G can be defined as $G = (V, E)$ where V is a non-empty finite **vertex set** and E is the **edge set**. [21, p.8] An edge e joins a pair of vertices in the graph together, defined as $e = (v_1, v_2)$ where $\forall(v_1 \wedge v_2) \in V$.

Definition of a Mesh

Let polygon mesh $P = (V, E, F)$, where V, E, F represents the set of vertices, edges, and faces respectively. In practice, polygonal meshes contain more components that influences surface appearance such as texture coordinates and vertex normals, however, the components described are sufficient for geometric processing. We assume vertex set and edge set of a mesh forms an **undirected simple graph**, where there is at most one edge for each pair of vertices and edges are bidirectional links.

A **mesh vertex** v is a 3D point of the form

$$\forall(x \wedge y \wedge z) \in \Re, v = (x, y, z).$$

The set of vertices is a point cloud representation of the geometry.

An **mesh edge** e is an unordered pair that connects two vertices. It is described in the form

$$\forall(v_1 \wedge v_2) \in V, e = \{v_1, v_2\}.$$

Vertices connected by edges form a wireframe of the geometry.

An **n-gon face** is formed from an arbitrary number of vertices

$$\forall(v_1 \wedge v_2 \wedge ... \wedge v_n) \in V, f_n = (v_1, v_2, ..., v_n),$$

however, we are only concerned with tri-faces f_3 when rendering, and quad-faces f_4 during content creation, where

$$\forall(v_1 \wedge v_2 \wedge v_3) \in V, f_3 = (v_1, v_2, v_3),$$

$$\forall(v_1 \wedge v_2 \wedge v_3 \wedge v_4) \in V, f_4 = (v_1, v_2, v_3, v_4).$$

A face describes the geometric surface of an object.

2.5.2 Topology

The term topology in mesh modelling refers to the organisation of mesh components. [15, 91] A mesh is a sample of the true object it represents, the construction of meshes can vary significantly even if they represent the same object. An intuitive explanation is to consider a high level of detail mesh with a low level of detail mesh of the same object - while visually the two might be evidently similar, this relationship is much harder to convey when observing the raw mesh data. The placement of vertices along with the arrangement of edges and faces determine how well the mesh will adapt to operations such as deforming and modifying algorithms, as well as how much computational resource it will consume.

Poles

A pole is a vertex that does not have exactly four edges connected to it. [15, p.92]. Meshes in production are organised using quad-faces, thus the majority of vertices are expected to have four edges. Poles with two or three edges are often necessary, however, poorly placed poles can be problematic when applying deformation.

Edge Loops

An edge loop is a set of connected non-forking edges that traverses the polygon geometry until it either forms an unbroken ring (cyclic edge loop), or ends at a pole. [15, p.93] Typically, edge loops are applied to mimic organic structures for maintaining a clean topology. [16, pp.10-12] Effective use of edge loops enable the mesh to deform smoothly during editing and animation. Edge loops can be extracted from the mesh edges, providing structural information of the geometry.

Chapter 3

Project Execution

This chapter discusses the steps undertaken to produce 3D hair geometry. A significant portion of this project is dedicated towards processing input polygonal hair mesh data so that the GP-LVM model can be effectively applied. First, we outline the process of acquiring appropriate 3D mesh input data and challenges imposed by using polygonal meshes as training data. Then, we address the data alignment problem, resolving it by feature extraction via approximating the data to obtain estimated parameters of a generative model. The latter sections of this chapter covers the training process and using output of the model to define geometry.

3.1 Training Data Set

3.1.1 Acquiring Training Data

The machine learning process begins with training data acquisition. Suitable 3D hair geometry is scarce when compared to alternatives such as images. Courtesy of Electronic Arts there exists an active community that produces free custom content for their gaming software - which includes polygonal hair. [11] Files acquired are encoded in the *Package* format, developed for a video game series, *The Sims*. Open-source community software *s4pe* is used to read the *Package* file and extract geometry in *Simgeom* format. [**?**] The *Simgeom* format is then converted to *OBJ* format using yet another open-source program *S4CASTools*. [**?**] The geometry extracted is already standardised in scale and orientation.

Figure 3.1: example polygon hair mesh training data.

3.1.2 Repairing Training Data

As mentioned, it is often the case for 3D mesh topology to be organised by quad-faces during production. Rendering pipelines convert quad meshes to triangles as an optimisation process. *Simgeom* data are triangulated meshes; reconstruction is performed on all input data to convert the geometry from triangulated meshes to have quad-face topology. It is possible to partially automate this process through functionality offered by the Blender API. A script to batch process conversion of multiple meshes executes the following steps in Blender:

1. First, a new blank scene is created.
2. Import a mesh object from the directory that contains the training data set.
3. Select the imported mesh and enter edit context mode which enables operations for manipulating objects.
4. Select all vertices of the mesh and apply the tri-to-quad conversion operation.
5. Enter object context mode to export the mesh.
6. Delete the mesh from the scene
7. Repeat from step 2 with the next mesh in the training directory until processing of all meshes.

Existing algorithmic solutions for converting a triangle mesh to a quad mesh are imperfect. Remaining triangle faces are required to be removed manually after the procedure. The conversion process alters the geometry very marginally but preserves the representation of hair structure. The quad-faced meshes are used for feature extraction and are assumed to be correct in the context that it is representative of a valid hairstyle.

Figure 3.2: (a) the original training mesh, (b) a partially retopologised mesh after tri-to-quad conversion operation, (c) a fully retopologised mesh after manually completing the conversion.

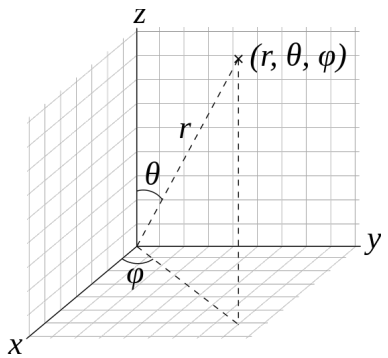
On average a input training mesh has X vertices and Y edges. The conversion algorithm on the training set used completes about ?% of the retopologising process. To retopologise the remaining triangles as quads manually, the hair mesh is separated into individual segments so that vertex, edge, and face components are not obstructed by layering geometry. On average, each mesh of the training set has around ? sub-meshes. Every sub-mesh must be inspected to find remaining trianglular polygons. There are many ways one can modify topology of a mesh, we assume that basic methods such as utilising the knife tool or dissolve tool of Blender takes only order of seconds per sub-mesh. The time for manually retopologising a partially converted mesh with the complexity of our training data takes order of hours each. Unfortunately, this meant that it is extremely time consuming to prepare meshes defined by the *Simgeom* format, thus for larger training sets we settled with including flawed examples.

3.2 Generative Model of Hair

To utilise machine learning methods for production of 3D hair geometry, we develop a generative model of 3D hair structure. Mesh data is tricky to compare as topology and fidelity alter both structure of the data encoded and dimensionality. We address the data alignment problem by approximating the mesh to obtain generative parameters of a similar output possible by a model. Training learns the relation of observed generative parameters obtained from approximating the mesh data.

Wang et al. (2009) parametrised a 3D scalp space using spherical projection. [20] This space approximated the surface of a scalp as a hemisphere with the centre denoting the origin. We apply a similar approach with a spherical coordinate system to determine the surface of a scalp.

The spherical polar coordinate system specifies a point from the origin using three parameters: (r, θ, ϕ). [1, pp.123-126] The **radial distance**, r, represents the distance from the origin. We also measure two angles: a **polar angle**, θ, from the positive z-axis (zenith direction) and an **azimuth angle**, ϕ, is measured from the xy-plane that is orthogonal to the zenith.



Source: https://commons.wikimedia.org/wiki/File:3D_Spherical.svg, public domain.

Figure 3.3: 3D spherical coordinate system

Uniformly placing points at a fixed radial distance and constant intervals within constrained angles give a collection of points that represent hair roots of the scalp. Increasing the number of hair roots allows sampling of higher fidelity. However, this also demands greater computational resources.

A total of 342 splines in our generative model is used to describe hair structure. We construct each spline from 10 points in a 3D space. The resulting feature vector is 10260 dimensions long.

Figure 3.4: Sphere meshes visualise root positions. The angle ranges specify coverage area of the scalp, while the interval of placements determines resolution. Fitting a sphere to the reference head mesh approximates the radial distance and origin.

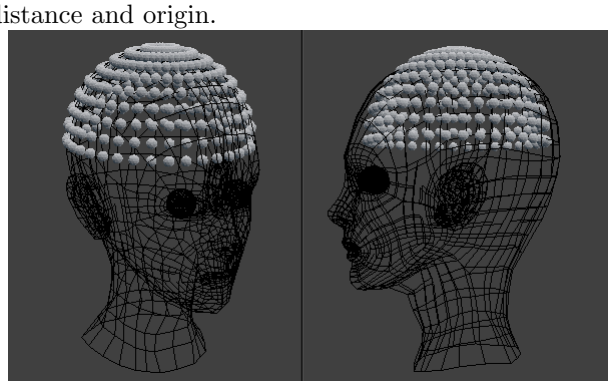


Figure 3.5: A possible configuration of 10260 parameters in the generative model, rendered on Plotly.

3.3 Approximating Generative Parameters from Input Data

3.3.1 Parsing OBJ File

Approximation begins by parsing the geometric data of OBJ files into a graph data structure of nodes and edges. The axis of exported OBJ files is different from the axis of Blender, to align the coordinate system, swap the Y and Z axis, then negate the Y axis. *OBJ* is a human-readable format that declares a mesh element per line. The elements we are concerned of are:

- geometric vertices, specified by a line that starts with the "v" character, followed by floating values that represent the x, y, z of the vertex position.
- face elements, specified by a line that starts with the "f" character, followed by a list of vertex indices that correspond to a vertex line.

Variations of the format may exist by implementation, but only the essentials identified are required for our purpose. An algorithm that parses *OBJ* files to retrieve a mesh graph and its corresponding vertex dictionary is given by:

Input: file name
Output: vertex dictionary of 3D points, mesh graph
for *line* ***in*** *file* **do**
  if *line starts with "v "* **then**
    parse vertex data
    add new vertex to vertex dictionary add new vertex to graph
  end
  else if *line starts with "f "* **then**
    parse face data
    extract edges from new face
    add edges to graph
  end
end
return vertex dictionary, graph

Algorithm 3.1: Parsing OBJ format

3.3.2 Spline Estimation

Edge Loop Extraction

The versatile structure of polygon meshes allows it to be an expressive representation, however, this flexibility can cause ambiguity when analysing geometric structure. We assume that the mesh only

contains sub-meshes of hair segments that have grid-like topology. Our assumption allows us to extract edge loops that describe the structure of hair segments. In the technical documentation of Blender, it describes an algorithm for edge loop selection. [**?**]:

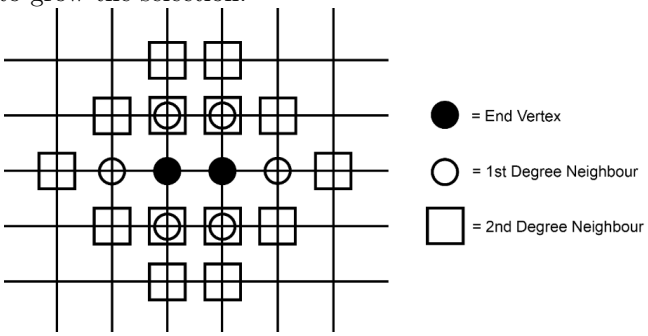
1. Given a starting edge, only continue searching adjacent edges if the candidates connect to exactly three other neighbours, as any other value would indicate either the border of a mesh or encountering a pole vertex.

2. Completing a cyclic edge loop ends the selection process.

3. Adjacent edges that share a face with the current one are discarded from consideration.

We devise an edge loop extraction algorithm that achieves the properties specified, illustrated in figure 3.6.

Figure 3.6: Our edge loop extraction algorithm begins by selecting an edge within the mesh graph. The two vertices of the edge are **end vertices**. We then proceed to *grow* the edge loop selection. Take the set of **first degree neighbours** of the *end vertices*, these nodes are the *candidates* for the edge loop. We remove edges that are part of the current edge loop from this set of first degree neighbours. We add the first degree neighbours to a set of face vertices for following iterations. Take the neighbours of the first degree neighbours as a set of **second degree neighbours**, removing its originating end vertex. A candidate node is only accepted to the edge loop if its set neighbours do not intersect with the set of face vertices. Accepted vertices are appended to the list of end vertices, this process is repeated until there are no end vertices left to grow the selection.



Boundary Edge Loops

Hair structure estimation begins by splitting the mesh into sub-meshes, determined by graph connectivity. For each segment mesh, we are interested in the edge loops that represent hair strands from the scalp roots. To find these edge loops, we must locate the boundary edge loops of the mesh and choose one to be the root edge loop.

Corner vertices are nodes that have exactly two edges, while border vertices have three edges. Boundary edge loops are determined by selecting an edge of a corner vertex and growing the edge loop. Any edge of a corner vertex will connect to a border vertex. The selection is specified to stop upon encountering another corner vertex, resulting in an edge loop of the bounds. We discard the vertices of the extracted boundary edge loop. Remove any corner vertices that no longer have any neighbours left in the set of border vertices. Repeat until there are no corner nodes left, thus successfully extracting the boundary of the mesh as a collection edge loops.

Root Edge Loop

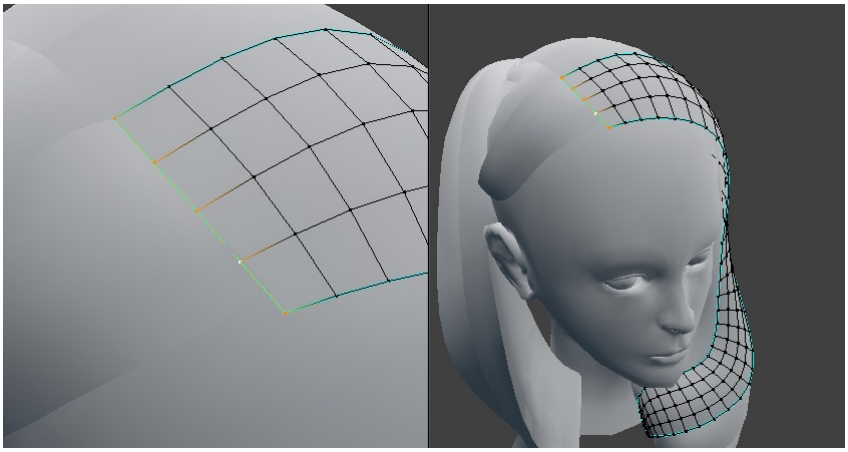
The *root edge loop* is the boundary edge loop that is closest to the surface of the scalp. It serves as a reference for where the hair strands begin. We determine the root loop by finding the average distance of each vertex in a boundary loop. Heuristically, the root border has minimal distance when aligned across the scalp.

There is a significant flaw in this approach as it will not correctly predict roots of hair segments that are represented by multiple meshes. A solution to this would be to accept hair that has clear roots first then iteratively join *floating segments* to the end of the closest corresponding *rooted segment* until there are no more, or the system is unable to connect any segments further.

```
while number of corner vertices greater than 0 do
    pick a corner vertex
    for vertex in neighbours of selected corner do
        if vertex in set of border or corner vertices then
            extract edge loops
            add edge loops to border edge loops collection
            remove vertices of extracted loops from border vertices
        end
    end
    for vertex in corner vertices do
        if all adjacent border edges of corner vertex have been removed then
            remove corner vertex
        end
    end
end
return border edge loops
```

Algorithm 3.2: Extracting boundary edge loops

Figure 3.7: Observe that the corner vertices have two edges, and the boundary edges have three. Determining the root edge loop is done by choosing the boundary edge loop that is closest to the scalp surface.



Algorithm 3.3: Determining root border edge loop

Pivotal Strand Splines

With the root loop, we can extract a collection of edge loops that represents the *pivotal* (descriptive) hair strands of the segment cluster. Given the neighbours of root vertices, removing the neighbours that are also root vertices will leave edges that represent key strands. When there is one edge, we take the edge loop as a pivotal strand representation. The edge loop will extend until it encounters a pole or ends on the boundary. It is useful to know where the strand starts (from the root), and where it ends. We extract a path from the edge loop graph by continually appending adjacent nodes, starting from the root node. The result forms a spline of connecting lines.

Algorithm 3.4: Extracting spline edge loops

Algorithm 3.5: Reconstructing spline path

Spline Repair Operation

A repair operator processes the strand splines to improve the approximation. First, the starting point of floating splines is attached to the nearest end of a rooted spline, only if there exists one within a specified vicinity. We discard the remaining floating splines that are not attached. Secondly, removing splines that are insignificantly short in length allows for more descriptive splines to be sampled.

Parametrising a Strand Spline

The procedure thus far returns a set of strand splines that have varying number of nodes. We produce a constant dimension through sampling to acquire a spline that is applicable for use as parameters of our generative model.

First establish intervals of the distance covered between the points of the original spline. Now suppose we want to evenly sample the spline with n points, we can compute the distance where the i_{th} sample point should travel along the spline as

$$distance = \frac{i \cdot s_l}{n},$$

where s_l is the spline length.

We determine the indices $[j, j+1]$ where the sample distance lies on the spline by comparing the sample distance to the distance of the interval list. The position, $\boldsymbol{P}$, of the sample point is

$$\boldsymbol{P} = \boldsymbol{S} + r\boldsymbol{D},$$

where $\boldsymbol{S}$ denotes the starting position that is the j_{th} point of the original spline, plus the unit vector direction to the next $j + 1_{th}$ point, multiplied by scalar r, the remaining distance to cover from the j_{th} interval.

A potential improvement for parametrising the strand splines is to concentrate sampling points at curves rather than evenly spaced. Sampling effectively allows the model to be descriptive with less parameters, mitigating problems introduced by the curse of dimensionality.

3.3.3 Structure Estimation

Spline estimation extracts an arbitrary number of splines. The next step is to choose a fixed number of splines for representing the hair geometry. We introduce a **selection operator** to associate a root with a spline such that the combination of splines are representative of the hair structure. Attributes that make a spline more desirable to a particular root include:

- *proximity* - how close the subject spline is to the root position.
- *significance* - how definitive a particular spline is, or the information conveyed by the subjected spline.
- *uniqueness* - whether there are other roots that have already captured the information conveyed by the subjected spline.

Several selection operators experimented are:

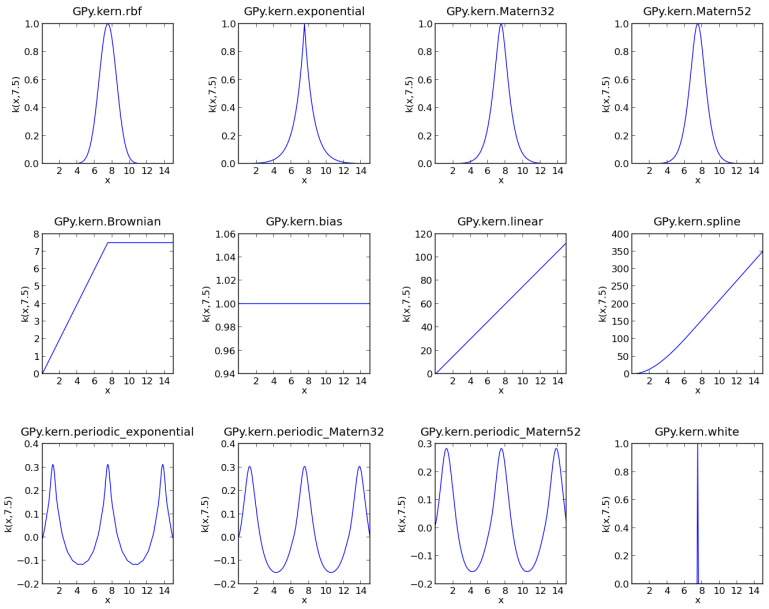
- *nearest strand selector* - it naively selects the closest spline of each root position.
- *average strand selector* - picks all strands within vicinity and takes the average representation.
- *unique selector* - considers all strands within vicinity and takes the spline that has maximises the minimum distance from any spline already selected by other roots.

Figure 3.8: Comparison of selection operators

3.4 Learning a Manifold with Bayesian GP-LVM

Once we have acquired a method to estimate the hair structure of the input data, we can then learn a latent manifold for generative parameters of hair.

A perspective that extends GP-LVM is the paradigm of **Kernel Method Based GP-LVMs**. [**?**] GP-PLVM can be equivalent to *Probabilistic Kernel Principal Component Analysis* (PKPCA) when replacing the $\boldsymbol{X}\boldsymbol{X}^T$ term of equation ? with a kernel matrix. The choice of a kernel influences performance of the model as it specifies the prior distribution for regression. Selection of the kernel depends on the task to be accomplished.



Source: GPy library documentation

Figure 3.9: GPy offers various standard kernels. Multi-modal kernels can be formed from combining kernels to represent data of complex nature.

The **radial basis function (RBF) kernel** models the data smoothly, interpolating between variables gradually. An **exponential kernel** places emphasis at peak likelihoods, changing very little from observed data when uncertainty is high. Using a **linear kernel** is equivalent to modelling with PPCA. Figure 3.10 illustrates the results of RBF, exponential, and linear kernels in our regression model.

Figure 3.10: As expected, a linearly embedded hairstyle is of little use as it can only extrapolate along one style.

Hyper-parameters of a kernel specifies its behaviour. A *lengthscale* parameter determines how much the distance of variables influence each other, and the *variance* describes its distribution. We apply *automatic relevance determination* (ARD), which uses the observed data to estimate the length-scale parameters.

The input data is transformed so that the origin is marginally above the origin of the scalp space. When the regression model is uncertain, it is sensible to bias the prediction towards the mean. Transforming the input data prevents generated hair to move inside the reference head mesh when uncertainty is high. An inverse transformation is applied on the output after prediction so that the generated mesh is located at its intended position.

3.5 Generation of Output

Completion of the training process provides a regression model that predicts hair structure from estimated input data using our generative model. A problem encountered was incompatibility of the plotting module

in the GPy library with the Python distribution of Blender, where a required standard module *TKInter* is not present. To overcome this issue, the training process plotted latent manifold is stored as an image and the regression model is serialised using the *Pickle* module of Python. A Blender add-on implementation can then load the latent manifold image on the UV image editor of the program and through the API, load the serialised regression model to predict latent variables sampled from the manifold.

Output generative parameters are used to create *guide splines* from the poly-line data structure of the Blender API. Polygon mesh geometry is produced from assigning these guide splines to a predefined taper curve and applied with the extrude functionality to create the appearance of hair geometry. An extension to our generative model would be specifying additional parameters such as orientation and width of hair segments, this would enable the regression model to also learn such attributes of the input data.

Figure 3.11: Blender Latent Hair Modelling Add-on

3.6 Logging User Activity

Our add-on implementation can be set to track user activity for acquiring behavioural data when using the latent manifold for geometry generation. Data logged include time spent exploring the manifold, mouse movement, and normalised likelihood of selected predictions. An output management procedure is developed to organise and write logged data automatically so that user testing can proceed seamlessly.

3.7 Project Management

Git is used for source control of the project implementation. The branching feature is useful for separating development of features. Maintaining multiple versions of the code base prevented issues caused by the interaction of incomplete features. The merging and rebasing tools helped conflict resolution. Descriptive atomic commits keep a log of progress and supports roll-back to older versions when necessary. A private repository backup was set up on a hosting service provider to prevent data loss and enable development on multiple machines with ease.

3.7.1 Time-line

Calender. Internal Deadlines.

foo	bar	baz
0	0	0

Table 3.1: This is an example table.

Issue tracking was performed on Trello.

Chapter 4

Critical Evaluation

4.1 Functional Testing

functional testing, including analysis and explanation of failure cases.
bald spots, inside head, intersection causes and solutions
3 pages

4.2 Behavioural Testing

behavioural testing, often including analysis of any results that draw some form of conclusion wrt. the aims and objectives
questionnaire analysis
3 pages

4.3 Statistical Analysis

null hypothesis, type 1-2 err, t-tests, empirical methods (see gmail), etc
1 pages

4.4 Evaluation

evaluation of options and decisions within the project, and/or a comparison with alternatives. 5 page

4.4.1 Training and Model

Assumes training data is correct. Triangles are a problem. Approximating loses a lot of data, data that is manually converted provides more information, giving better output visually.

Works better when styles are similar, otherwise strands interpolate in between to be incorrect such as inside head.

4.4.2 Choice of Learning Model

Linear, decision trees, neural nets, GPLVM Supervised, unsupervised, genetic

4.4.3 Comparing with Other methods

AutoHair, Exemplar, etc

4.4.4 Extending to other models

Simulating foliage, grass. Can easy do surface Combine models Deep GPs 2 pages

Addressing Non-Intuitive Use

Plot small image of output in corresponding area of latent variable model to illustrate the certain areas (training data) so users can explore around it for variation.

Observations

What I can do: transferring to fixed objects like a face: mouth, eye, etc that is more "predictable".

4.5 Applications

MMOs, games, etc. 3D latent space with VR. 2 page

Chapter 5

Conclusion

5.1 Summary

(Re)summarise the main contributions and achievements, in essence summing up the content. 2 pages

5.2 Project Status

Clearly state the current project status (e.g., "X is working, Y is not") and evaluate what has been achieved with respect to the initial aims and objectives (e.g., "I completed aim X outlined previously, the evidence for this is within Chapter Y"). There is no problem including aims which were not completed, but it is important to evaluate and/or justify why this is the case. 2 pages

5.3 Future Work

Outline any open problems or future plans. Rather than treat this only as an exercise in what you *could* have done given more time, try to focus on any unexplored options or interesting outcomes (e.g., "my experiment for X gave counter-intuitive results, this could be because Y and would form an interesting area for further study" or "users found feature Z of my software difficult to use, which is obvious in hindsight but not during at design stage; to resolve this, I could clearly apply the technique of Smith [7]"). 1 page

Bibliography

[1] G. B. Arfken and H. J. Weber. *Mathematical Methods for Physicists*. Academic Press, 6th edition, 2005.

[2] B. G. Baumgart. *Winged Edge Polyhedron Representation*. Advanced Research Projects Agency. AD-755 141., 1972.

[3] W. Baxter and K. Anjyo. *Latent Doodle Space*. Eurographics 2006. Vol. 25, 3., 2006.

[4] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[5] N. D. Campbell and J. Kautz. *Learning a manifold of fonts*. ACM Transactions on Graphics (TOG) 33, 4, 91., 2014.

[6] Shaq T. Wu H. Weng Y. Chai, M. and K Zhou. *AutoHair: Fully automatic hair modeling from a single image*. ACM Trans. Graph. 35, 4 (July), 116:1116:12, 2016.

[7] F. Bevilacqua D. M. D. Carli, C. T. Pozzer and M. C. d Ornellas. *A survey of procedural content generation techniques suitable to game development*. X Simposio Brasileiro de Games e Entretenimento Digital, 2011.

[8] R. Milo. et al. *Number of hairs on human head*. BioNumbers. BNID 101509, 2017.

[9] K. et al. Grochow. *Style-Based Inverse Kinematics*. SIGGRAPH '04 ACM SIGGRAPH 2004 Papers. pp.522-531, 2004.

[10] [PCA 1901] http://stat.smmu.edu.cn/history/pearson1901.pdf.

[11] [TSR] https://www.thesimsresource.com/.

[12] I. T. Jolliffe. *Principal Component Analysis*. Springer Series in Statistics, Springer-Verlag, 2nd edition, 2002.

[13] N. D. Lawrence. *Probabilistic non-linear principal component analysis with Gaussian process latent variable models*. The Journal of Machine Learning Research 6, 17831816, 2004.

[14] A. et al. Limpaecher. *Real-time Drawing Assistance through Crowdsourcing*. ACM Trans. Graph. 32, 4 (July), 2013.

[15] T. Mullen and C. Andaur. *Blender Studio Projects: Digital Movie-Making*. Wiley Publishing, Inc, 2010.

[16] B. Raitt and G. Minter. *Digital Sculpture Techniques: How to apply the principles of traditional sculpture to make stunning 3D characters*. Nichimen Graphics, 2000.

[17] C. E. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[18] M. E. Tipping and C. M. Bishop. *Probabilistic Principal Component Analysis*. Microsoft Research, Cambridge, UK, 1998.

[19] M. K. Titsias and N. D. Lawrence. *Bayesian Gaussian Process Latent Variable Model*. AISTATS. Vol. 9, 2010.

[20] L. et al. Wang. *Example-Based Hair Geometry Synthesis*. ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH 2009. Vol. 28, Issue 3., 2009.

[21] R. J. Wilson. *Introduction to Graph Theory*. Prentice Hall, 4th edition, 1996.

Appendix A

An Example Appendix

Content which is not central to, but may enhance the dissertation can be included in one or more appendices; examples include, but are not limited to

- lengthy mathematical proofs, numerical or graphical results which are summarised in the main body,
- sample or example calculations, and
- results of user studies or questionnaires.

Note that in line with most research conferences, the marking panel is not obliged to read such appendices.