University of
BRISTOL

DEPARTMENT OF COMPUTER SCIENCE

ART-CG:

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

Dillon Keith Diep

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.

Saturday 6$^{\text{th}}$ May, 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, Saturday 6th May, 2017

Contents

List of Figures

List of Tables

List of Algorithms

List of Listings

Executive Summary

Production of 3D virtual worlds is a time-consuming and costly process that also demand knowledge. 3D assets encompass a vast range of applications, ranging from simulations and research to contributing towards the functioning of many industries. Our proposition of probabilistic non-linear dimension reduction consolidates observed variables to provide a parsimonious collection of latent variables, drastically diminishing the complexity of the production pipeline. Simplifying a procedure within the pipeline yield increased throughput and offer an opportunity for rapid prototyping. A non-linear approach minimises loss of information; thus high dimensional data can be reduced effectively to dimensionality that is sufficiently low. Non-experts may find directing a few descriptive components within a low dimension representation easier than controlling an overwhelming number of attributes present in the original data.

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. The properties of 3D hair geometry pose significant challenges for machine learning solutions. Machine learning typically uses 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 faces 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 to 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 content faster. However, the lack of control over the final result causes standard automation to be typically less desirable than traditional methods of 3D modelling. The research hypothesis of this study is that of applying non-linear probabilistic dimensionality reduction to improve the efficacy of creative content production for exceptionally high dimensional data such as complex 3D hair geometry on virtual humanoids.

Deliverables:

- Formulating a generative model for 3D humanoid hair structure.
- Resolving the alignment problem by repairing raw input mesh data and feature extraction by approximating parameters of our generative model.
- Learning a low dimension latent-space of high dimensional hair structure data.
- Investigated the performance of various kernels and its impact on the latent variable model.
- Implemented an add-on package for a 3D production program, Blender.
  - Generating guiding hair strands from the output of our regression model.
  - Developed with a small training set that is practical for content creators.
  - Achieving real-time performance, matching state of the art non-learning tools.
  - Including functionality to records user activity for analysis.
- Evaluation of data collected from survey and tests.

Supporting Technologies

- Python with standard libraries for implementation.
- GPy library to implement the regression model.
- NumPy, SciPy, and Networkx for operations and processing meshes.
- PIL, Matplotlib, and Plotly for plotting images and graphs.
- Pickle for serialising objects as files.
- Blender API for mesh processing and the Blender 3D program for demonstration of our regression model.
- Google Forms to create feedback survey.

Notation and Acronyms

x	:	Scalar
$\boldsymbol{x}$	:	Column vector
$\boldsymbol{x}^T$	:	Row vector, superscript T denotes transpose
x_i	:	the i-th value of a vector $\boldsymbol{x}$
$\boldsymbol{I}$	:	Identity matrix
$\boldsymbol{X} \in \Re^{n \times m}$	:	Matrix of real numbers with size $n \times m$
$\mathcal{N} \sim (\mu, \sigma^2)$	:	Gaussian (normal) distribution with mean μ and variance σ^2.
DR	:	Dimensionality Reduction
PCA	:	Principal Component Analysis
GP-LVM	:	Gaussian Process Latent Variable Model

Acknowledgements

I would like to thank coffee

Chapter 1

Contextual Background

This dissertation explores assisted real-time content generation through training regression models using unsupervised machine learning for the production of 3D hair geometry. The purpose of a 3D object is to describe a surface. Typically we define geometric surfaces using a combination of many primitive components. The craft of 3D content is a meticulous procedure as content creators are required to work with enormously high dimensional observed data. The objective of this project is to demonstrate the potential of probabilistic non-linear dimensionality reduction on creative content production by presenting a framework that improves the efficacy of producing 3D hair geometry. In this chapter, we will introduce complications of producing 3D hair geometry with traditional methods and past research on machine learning applied to creative fields. We will also outline the motivation and significance for research, including challenges and central objectives of this study.

1.1   Production of 3D Content

There are multiple representations of 3D objects in computer graphics. One way to obtain 3D geometry data is to sample surfaces of the physical world with a 3D scanner. Common representations of sampled geometric data include point clouds, range maps, and voxels. Such representations are effective for sampling as they reconstruct a surface from allocating many simple components.

Figure 1.1: 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.

Representations have advantages and disadvantages depending on the use case. It is possible to convert between representations; however, converting between representations may incur data loss. Precise representations are often mathematical models such as *NURBS* (Non-Uniform Rational Basis Spline) - frequently applied in the *CAD* (Computer-Aided Design) industry. Mathematical models are precise as they do not suffer from floating-point inaccuracy that is present in many representations. The most widely applied representation for *CGI* (Computer-Generated Imagery) is *polygonal meshes*. In a production environment, it is preferred to define geometry specifically to requirements of the design as opposed to capturing examples. Polygon meshes are simple to define, yet with established techniques such as UV texturing and normal mapping, are sufficiently expressive for visual purposes. A significant amount of study has been conducted in the field of polygon meshes to improve its versatility. The study of polygonal meshes is deeply rooted in computer graphics.

The elements of a polygonal mesh are *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 performing algorithmic operations on a mesh. In practice, professionals create polygon meshes with majority quad-face topology (faces formed from four edges) during production. The rendering pipeline often automatically converts polygon meshes to triangle faces (formed from three edges) as an

optimisation process. *Quadrilateral mesh* form ring loops that we call *edge loops* which can be used to define the structure of geometry, thus conform better with editing tools and algorithms than *triangle meshes*.

State of the art 3D production software such as AutoDesk Maya, 3DS Max, and Blender are advanced programs with a 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 high learning curve with an overwhelming user experience is not beginner-friendly. In fact, the most functionality is only accessible through an API (Application Program Interface) via a scripting language. Experienced users welcome this complexity as it allows creators to customise their ideal workflow, but it also creates a high barrier to entry for non-experts who wish to possess the capability of such 3D programs.

1.1.1   3D Hair Geometry

In recent years, leading 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, and polygon hair can flexibly represent different art styles. In some cases, a blend of multiple representations is used to balance between cost and quality. 3D hair in cinematography with a large budget can afford to render hair with much higher fidelity for major characters, but would still consider using efficient variants for scenarios such as crowd simulation. Ultimately, we can observe that the representation of virtual hair follows a structure of splines with control points that define the overall organisation of strands or segments. This information will allow transferral between representations to an extent.

On average, a human is born with between 90,000 to 150,000 scalp hair follicles [12]. It is computationally very expensive to render and animate physically correct hair, but creative liberties are taken to approximate or stylise 3D hair such that it is both acceptable aesthetically and feasible regarding performance. This study considers modelling of hair geometry, the motion of hair is assumed to be its default resting pose.

To add

Figure 1.2: Image of hair geometry.

There are various ways to construct a polygonal hair mesh. We are concerned with hair meshes formed from multiple planar sub-meshes of hair locks or segments. This type of hair mesh can express a variety of styles but produce densely arranged geometry. Producing hair mesh data is challenging as content creators manipulate data that is exceptionally high dimensional, this reflects on **the curse of dimensionality**, 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. There are two perspectives where the curse of dimensionality applies in our project. First, from the viewpoint of a 3D content creator, appending data points to geometry on a 3D program becomes increasingly difficult. Tightly positioned components are laborious for creators to find or select, with each action having less impact on the overall surface. The other perspective is that of algorithmic complexity. Machine learning methods scale and generalise better when using a few key features, use of many features is computationally expensive and risks overfitting.

Procedural generation techniques produce output that adheres to rules established by the generative model defined. Generation of terrains and city modelling sometimes employ procedural techniques to take advantage of its systematic nature to mass produce variations in agreement with specified patterns [11]. Fractals and methods such as the Lindenmayer system have been used to create patterns that resemble those observed in nature [**?**]. Automated techniques such as the ones discussed, however, are seldom used for modelling distinct objects with a specific design. It is an involved process to control the output of procedurally generated content without heavily restricting its capabilities. Automated methods that do not learn cannot adapt to changing demands without reimplementation.

1.2   Related Research of Machine Learning in Creative Fields

Three major paradigms divide the task of machine learning:

- *Supervised learning* is provided input training examples with desired outputs to learn the mapping of inputs to an output.

- *Unsupervised learning* seeks to learn the structure of and relation between input data.

- *Reinforcement learning* iteratively improve a pool of solutions by simulating an environment that applies 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. In machine learning, prediction of continuous variables is a **regression** problem.

Style-based inverse kinematics introduced the Scaled Gaussian Process Latent Variable Model to learn the probabilistic distribution of a 3D human posture model [13]. 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.

A latent doodle space is the use of a low-dimension latent space that has been applied on simple line drawings [7]. 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.

A study by Campbell & Kautz presented a framework that learns the latent manifold of existing font styles [9]. 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.

Drawing assistance powered by large-scale crowd-sourcing explored the potential of data driven drawing to prompt for correction by achieving an artistic consensus [18]. 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.

Chai et al. introduced AutoHair, a method for automatic modelling of 3D hair from a portrait image [10]. 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.

Research regarding creative content often parametrise the input data so that machine learning is applicable. There is no clearly defined solution for a problem in the creative field, effective solutions strive for versatility, employing consensus decision making or offering multiple solutions. To overcome the challenges introduced, dimensionality reduction through unsupervised learning with probabilistic latent variable models such as the Gaussian Process Latent Variable Model (GP-LVM) [17] present an opportunity to learn stylistic properties of design and predict multiple acceptable outputs by analysing the likelihood.

1.3 Motivation and Significance

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 creation. In comparison to other forms of surfaces, hair meshes are densely concentrated, containing many data points that are exhausting to edit. Maintaining topology and issues such as overlapping surfaces are problematic among detailed meshes. 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. Each alteration makes less impact as the geometry becomes more detailed and well-defined. The space of sensible changes become smaller. Design and production of 3D geometry remain a slow and delicate process.

Non-linear dimensionality reduction models are attractive candidates to resolve difficulties imposed by high dimensional data such as 3D polygonal hair geometry. That said, the process of machine learning is more involved than simply inputting training data into a model. In practice, data acquired are frequently encoded in complicated structures. Such data is seldom useful in its original form; feature extraction may be required to find an orderly representation that helps facilitate the learning stage. 3D meshes are complicated data structures that are heavily ambiguous when interpreting its data values without visualising the geometry. A motivational point for this project is to demonstrate how dimensionality reduction can be employed to tackle a real world problem such as content production.

Learning the relation of hair structure allows the potential of discovering new hairstyles. It can also be used to output base geometry that fits the target output better than existing geometry available. Generative methods could ensure a level of quality, a clean topology that fits established specifications. The application of machine-learning based tools 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 systems such as probabilistic regression models to prevent inappropriate or undesirable creation from being produced while providing options that surpass existing alternatives. Assisted content generation using machine learning can provide a convenient, non-intrusive and intuitive method for rapidly generating new hair geometry from existing data.

1.4 Challenges

This study faces multiple challenges. Firstly, 3D meshes are difficult to compare. The training data in its raw form will have varying dimensions. We can view meshes as samples of the actual surface. Thus meshes that represent the same object could differ drastically in the 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 standard size of readily available training data is much smaller. Public repositories of 3D polygonal hair ordinarily contain up to thousands of meshes [4]. Studios that store and organise past production may match the extent 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 the 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.

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

fast in presenting observable output.

1.5 Central Objectives

- Resolving the alignment problem of 3D data through a representative generative model.

- Explore the application of non-linear dimension reduction for high dimensional data such as 3D hair geometry.

- Investigate the use of latent variables for identifying stylistic properties of 3D content.

- Demonstrate the use of a non-linear manifold to generate new hairstyles from training data.

- Enable an intuitive method for non-experts to easily create 3D hair geometry.

- Observable output demand performance comparable to real-time for practical use.

Chapter 2

Technical Background

In the previous chapter we briefly introduced the *curse of dimensionality*. High dimensional data is often counter-intuitive to perceive and process. Unfortunately, it is common for observed data to be in a representation of greater dimensionality than it requires. This gives rise to the notion of **dimensionality reduction**, a sub-field of machine learning that is motivated find a descriptive low-dimensional representation of data. This chapter will establish the technical foundation required for understanding the probabilistic non-linear dimensionality reduction model we use for assisted real-time content generation of polygonal hair. Starting with principal component analysis (PCA), we review its probabilistic equivalent that contributes to the derivation of the Gaussian Process Latent Variable Model (GP-LVM). Then, we give a formal definition of polygon meshes from the perspective of graph theory - along with concepts that are useful for mesh processing.

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 [15], and independently developed by Hotelling [14], where the standard algebraic derivation of PCA was presented.

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 $\boldsymbol{\alpha}_1 = [\alpha_{11}, \alpha_{12}, ..., \alpha_{1n}]$ is a vector of n constants such that [16, p.4]:

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

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 design matrix $\boldsymbol{X}$ [16, pp.44-46],

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

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.

A limitation of standard PCA is the lack of a probabilistic solution. It is very likely for there to be a multiple sets of principal components that is equally acceptable. In the case of polygonal hair production, this aspect is more prominent as an artistic environment desires to evaluate various plausible designs. A probabilistic model will enable exploration of other stylistic embeddings for hair structure. One way to formulate a probabilistic model is to introduce the notion of noise among observed variables.

2.2 Multivariate 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 normally distributed [8, 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 [8, p.78]:

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

$$\mathcal{N}(y|\mu, \sigma^2) \equiv p(y|\mu, \sigma^2).$$

For a d-dimensional vector $\boldsymbol{y}$, the multivariate Gaussian distribution with mean vector d-dimensional $\boldsymbol{u}$, $d \times d$ covariance matrix $\boldsymbol{\Sigma}$, and the determinant of $\boldsymbol{\Sigma}$ as $|\boldsymbol{\Sigma}|$, is

$$\mathcal{N}(\boldsymbol{y}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{\frac{d}{2}}\sqrt{|\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right).$$

The Gaussian distribution is *closed* under addition (2.1), scaling (2.2), and multiplication (2.3) - all of which yields a result that is also a Gaussian distribution [21, p.200]. These notable properties will be useful in later sections as it allows analytical integration of multivariate Gaussian distributions.

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

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

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

$$Z^{-1} = \frac{1}{(2\pi)^{\frac{d}{2}}\sqrt{|\boldsymbol{A}+\boldsymbol{B}|}} \exp\left(-\frac{1}{2}(\boldsymbol{a}-\boldsymbol{b})^T(\boldsymbol{A}+\boldsymbol{B})^T(\boldsymbol{a}-\boldsymbol{b})\right),$$

$$\boldsymbol{c} = \boldsymbol{C}(\boldsymbol{A}^{-1}\boldsymbol{a} + \boldsymbol{B}^{-1}\boldsymbol{b}),$$

$$\boldsymbol{C} = (\boldsymbol{A}^{-1} + \boldsymbol{B}^{-1})^{-1}).$$

Let $w \sim \mathcal{N}(\mu_1, \sigma_1^2)$ and $h \sim \mathcal{N}(\mu_2, \sigma_2^2)$ 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

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

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

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

where $\boldsymbol{D} \in \Re^{n\times n}$ is the diagonal matrix of the variances [8, p.78].

2.3   A Probabilistic Model for PCA

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* [22].

A *latent variable model* transforms a set of n d-dimensional observed variables encoded as a design matrix, $\boldsymbol{Y} = [\boldsymbol{y}_1, ..., \boldsymbol{y}_n]^T$, to a set of n q-dimensional latent (unobserved) variables, $\boldsymbol{X} = [\boldsymbol{x}_1, ..., \boldsymbol{x}_n]^T$. 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. For each observed data point, $\boldsymbol{y}_i \in \boldsymbol{Y}$, $1 \le i \le n$, there is an associated latent variable $\boldsymbol{x_i}$. The original data can be represented in terms of the corresponding latent variable as

$$\boldsymbol{y}_i = \boldsymbol{W}\boldsymbol{x}_i + \boldsymbol{\mu}_i + \boldsymbol{\epsilon}_i. \tag{2.5}$$

The matrix $\boldsymbol{W}$ represents the linear relationship between the latent-space with the data-space. The parameter $\boldsymbol{\mu}_i$ allows for non-zero mean, and the $\boldsymbol{\epsilon}_i$ 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. PCA reduce dimensionality without explaining the relation between the original data and its principal components. Regression is possible with a model that explains the relation between observed variables and latent variables.

Figure 2.1: In a latent variable model, a latent variable, $\boldsymbol{X}$, is input into a function, $\boldsymbol{F}$, which maps to the output observed variables, $\boldsymbol{Y}$.
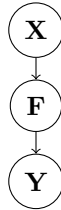
The latent variable model developed by Tipping and Bishop performs principal component analysis by modelling the noise parameter of equation 2.5 as an isotropic, spherical Gaussian distribution The noise values, $\epsilon_i \in \Re^{d\times 1}$, are sampled from a independent spherical Gaussian distribution,

$$\boldsymbol{\epsilon}_i \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\beta}^{-1}\boldsymbol{I}).$$

The conditional probability distribution of a observed variables $\boldsymbol{y}_i$ given input variables $\boldsymbol{x}_i$ is thus Gaussian distributed as

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

The prior of latent variables is assumed to be standard Gaussian with zero mean and unit covariance, defined as $\boldsymbol{x}_i \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. 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 $\boldsymbol{R}^T$ can be applied to the basis, forming the correlated Gaussian,

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

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, with a prior as standard Gaussian, $\mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$,

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

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

where the observation covariance model is $\boldsymbol{C} = \boldsymbol{W}\boldsymbol{W}^T\beta^{-1}\boldsymbol{I}$, with corresponding log-likelihood [17]

$$\mathcal{L} = \frac{n}{2}(d\,ln(2\pi) + ln|\boldsymbol{C}| + tr(\boldsymbol{C}^{-1}\boldsymbol{S})), \tag{2.6}$$

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

We can write the likelihood for a data point as

$$p(\boldsymbol{y}_i|\boldsymbol{x}_i, \boldsymbol{W}, \boldsymbol{\beta}) = \mathcal{N}(\boldsymbol{y}_i|\boldsymbol{W}\boldsymbol{x}_i, \boldsymbol{\beta}^{-1}\boldsymbol{I}). \tag{2.7}$$

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, $p(\boldsymbol{x_i}) = \mathcal{N}(\boldsymbol{x_i}|\boldsymbol{0}, \boldsymbol{I})$, marginalisation of the integral obtains the marginal likelihood of each data point as

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

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

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

The Principal Subspace of PPCA

Tipping and Bishop showed that all potential solutions for $\boldsymbol{W}$, the likelihood (2.6), is of the form [22]

$$\boldsymbol{W} = \boldsymbol{U}_q(\boldsymbol{K}_q - \sigma^2\boldsymbol{I})^{\frac{1}{2}}\boldsymbol{R}.$$

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

$$\boldsymbol{W}_{ML} = \boldsymbol{U}_q\boldsymbol{L}\boldsymbol{R}, \tag{2.8}$$

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

The matrix $\boldsymbol{U}_q$ contains the column vectors that are the principal eigenvectors, $\boldsymbol{\Lambda}_q = [\lambda_1, ..., \lambda_q]$ represents the diagonal matrix of the corresponding eigenvalues, and $\boldsymbol{R}$ represent an arbitrary orthogonal rotation matrix. Maximising the likelihood of $\boldsymbol{W}$ by equation 2.8 on the latent variable model defined by equation 2.5 maps the latent-space to the principal subspace of the observed data. Satisfying $\boldsymbol{W}_{ML}$, the latent variable model is effectively equivalent to standard principal component analysis.

2.4 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) [17]. A non-linear embedding is more suitable model for capturing high dimensional data such as polygonal hair structure.

2.4.1 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 (2.7) 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

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

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

$$p(\boldsymbol{y}_{:,i}|\boldsymbol{X}, \boldsymbol{\beta}) = \mathcal{N}(\boldsymbol{y}_{:,i}|\boldsymbol{0}, \boldsymbol{X}\boldsymbol{X}^T + \boldsymbol{\beta}^{-1}\boldsymbol{I}). \tag{2.9}$$

The objective function is the log-likelihood

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

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

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 a matrix whose orthonormal column vectors are the first eigenvectors of $\boldsymbol{Y}\boldsymbol{Y}^T$. $\boldsymbol{L}$ is a diagonal matrix of singular values, whose i^{th} element is $l_i = (\lambda_i - \frac{1}{\beta})^{-\frac{1}{2}}$, where λ_i is the eigenvalue associated with the ith eigenvector $\frac{1}{d}\boldsymbol{Y}\boldsymbol{Y}^T$. $\boldsymbol{V}$ is an arbitrary rotation matrix. Lawrence showed that the eigenvalue problem developed here is equivalent to the eigenvalue problem solved in probabilistic PCA, and thus, dual probabilistic PCA 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.

2.4.2 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. A prior probability distribution is an assumption of belief before taking into account of evidence. Observing the output of a continuous function provides information regarding its behaviour around that specific point. For a noiseless model, we can be certain that the input must intersect the point at observed output. This refines our belief to obtain a posterior. A model with noise can *infer* that the mapping is nearby the observation, thus treating each observed variable as an independent distribution allows us to take into account of the noise model for prediction.
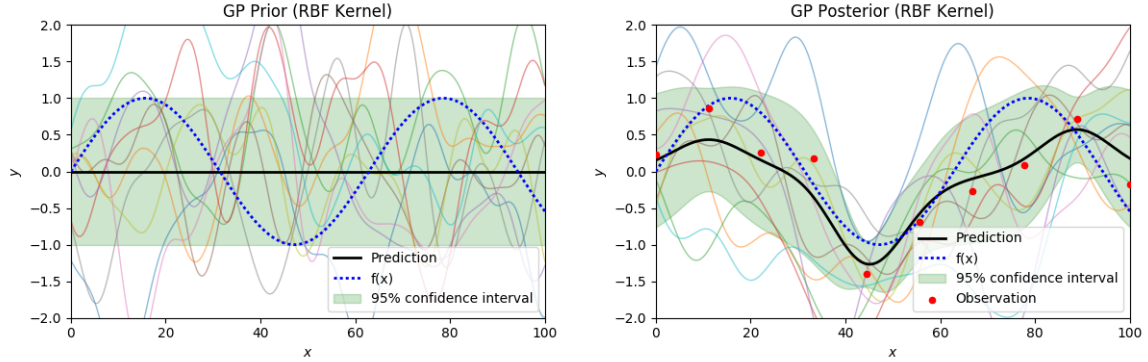
Figure 2.2: To the left is a prior distribution of a Gaussian process, and to the right is the posterior distribution after updating with observations that are corrupted by noise. The semi-transparent functions represent samples from the likelihood distribution.

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 $f(\boldsymbol{x})$ [21, 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.

Gaussian Processes 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,

$$f(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{w} \qquad y = f(\boldsymbol{x}) + \epsilon.$$

Here, $\boldsymbol{w}$ represents the parameters that specify the behaviour of the function. Take $\boldsymbol{X}$ as the design matrix of the input values. 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,

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

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

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

The marginal likelihood is given by

$$p(\boldsymbol{y}|\boldsymbol{X}) = \int p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{w})p(\boldsymbol{w})d\boldsymbol{w}.$$

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 [17], $\epsilon \sim \mathcal{N}(\boldsymbol{0}, \beta^{-1}\boldsymbol{I})$. The covariance function (kernel) is thus,

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i^T \boldsymbol{x}_j + \beta^{-1}\delta_{ij}. \tag{2.11}$$

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

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

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 (2.11). Thus, the marginal likelihood of dual probabilistic PCA is a product of d independent Gaussian processes. The covariance function of a Gaussian process describes the properties of functions, such as variability. Learning in Gaussian processes is to determine hyper-parameters of a covariance function that is suitable for the problem being modelled.

2.5   Bayesian Gaussian Process Latent Variable Model

The Bayesian Gaussian Process Latent Variable Model (Bayesian GP-LVM) [23] extends GP-LVM. The standard GP-LVM method trains by finding the *maximum a posteriori* (MAP) estimate of $\boldsymbol{X}$ and jointly maximizing with respect to the hyper-parameters. 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 hyper-parameters.

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. A variational distribution, $q(\boldsymbol{X})$, can instead be used to approximate the posterior distribution over the latent variables, $p(\boldsymbol{X}|\boldsymbol{Y})$.

$$q(\boldsymbol{X}) = \prod_{i=1}^{n} \mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\mu}_i, \boldsymbol{S}_i).$$

$\boldsymbol{\mu_n}$ and $\boldsymbol{S}_i$ are the variational parameters. $\boldsymbol{S_i}$ is taken as a diagonal covariance matrix. The variational distribution can then be used to obtain a Jensen's lower bound on $\log p(\boldsymbol{Y})$:

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

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

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

The $KL(q||p)$ term is the negative KL divergence between the variational posterior distribution $q(\boldsymbol{X})$ and the prior distribution $p(\boldsymbol{X})$ over the latent variables. As the KL divergence is Gaussian, by the closed properties of the Gaussian distribution, we know it is analytically tractable. The problematic term is $\tilde{F}(q)$, which is solved by variational sparse Gaussian process regression [23].

A fully marginalised GP-LVM approximated by variational inference establishes a Bayesian perspective that is robust to overfitting, thus applicable to data sets even with missing or uncertain observed data. Now that we have established a foundation for probabilistic latent variable models, the next section will look at the data structure of our training input data, the 3D polygon mesh representation.

2.6 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 [6] 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.

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** [25, p.8]. An edge e joins a pair of vertices in the graph together, defined as $e = (v_1, v_2)$ where $\forall (v_1 \wedge v_2) \in V$.

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.

A **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

$$\forall (v_1 \wedge v_2 \wedge v_3) \in V, f_3 = (v_1, v_2, v_3),$$
$$\forall (v_1 \wedge v_2 \wedge v_3 \wedge v_4) \in V, f_4 = (v_1, v_2, v_3, v_4).$$

The face component describes the geometric surface of an object.

2.6.1 Topology

The term topology in mesh modelling refers to the organisation of mesh components [19, p.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 [19, 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 [19, p.93]. Typically, edge loops are applied to mimic organic structures for maintaining a clean topology [20, 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.

2.7   Summary of Technical Background

Our motivation for dimensionality reduction on 3D geometry with a latent variable model aspires to simplify the production process by explaining the underlying behaviour of observed mesh data. The learning models discussed in this chapter adopt a probabilistic framework so that artists can retain creative control. The next chapter will examine dimensionality reduction for the mesh data structure with the learning models introduced.

Chapter 3

Project Execution

This chapter covers the process of assisted real-time content generation for 3D hair geometry by applying non-linear dimensionality reduction to obtain a regression model and a latent manifold for selecting output. A significant portion of this project is dedicated towards processing input polygonal hair mesh data so that the non-linear dimensionality reduction 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 discuss why it is difficult to apply machine learning methods on mesh data structure and presenting a solution to address this issue. The latter sections of this chapter covers the training process and using output of the regression model to define geometry.

3.1 Training Data Set

The machine learning process begins with training data acquisition. Suitable 3D hair geometry is scarce when compared to other mediums such as images. Courtesy of Electronic Arts there exists an active community that produces free for non-commercial use custom content for their gaming software - which includes polygonal hair [4]. 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 [3]. The *Simgeom* format is then converted to *OBJ* format using yet another open-source program *S4CASTools* [2]. The geometry extracted is already standardised in scale and orientation.
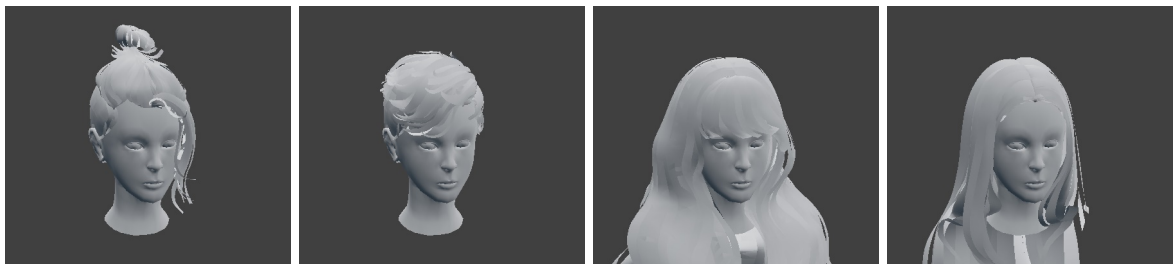
Figure 3.1: examples of polygon hair mesh training data.

3.1.1 Retopologising Training Data

Retopology is the act of refactoring the topology of a surface into a different arrangement of mesh components. As mentioned, it is often the case for 3D mesh topology to be organised by quadrilateral 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 quadrilateral 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.
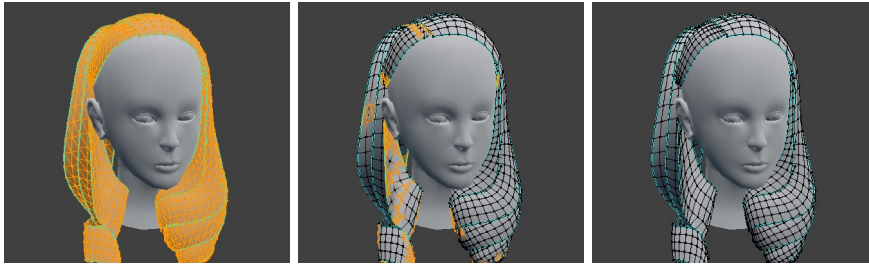
Figure 3.2: Triangular faces are highlighted using a selection procedure for all faces that are made of exactly three edges. To the left is an original mesh acquired, the middle shows a selection of remaining triangles after the automatic tri-to-quad conversion, and the right displays a retopologised mesh of entirely quads.

Sampling four input mesh from the training set (those displayed in figure 3.1), on average there are around 2×10^2 sub-meshes, 1×10^4 vertices and 3×10^4 edges per mesh. To retopologise the remaining triangles as quads manually, the hair mesh is separated into individual segments so that visibility of layering sub-mesh surfaces can be toggled to not obstruct vertex, edge, and face components. Each 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 a non-expert 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 regression models with larger training sets we settled with including partial 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. To successfully learn the relation of mesh data, we must first standardise the 3D representation by approximating the mesh to sample generative input of a similar representation possible by a model. Training learns the relation of observed generative inputs obtained from approximating the mesh data.

Wang et al. (2009) parametrised a 3D scalp space using spherical projection [24]. 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) [5, 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.
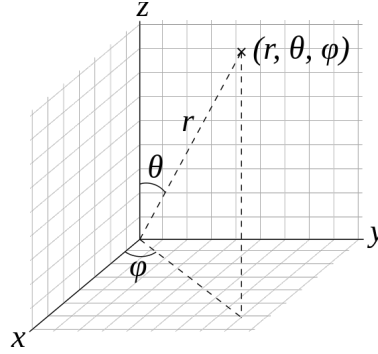
Figure 3.3: Source: https://commons.wikimedia.org/wiki/File:3D_Spherical.svg, Public Domain. The 3D spherical coordinate system.

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.

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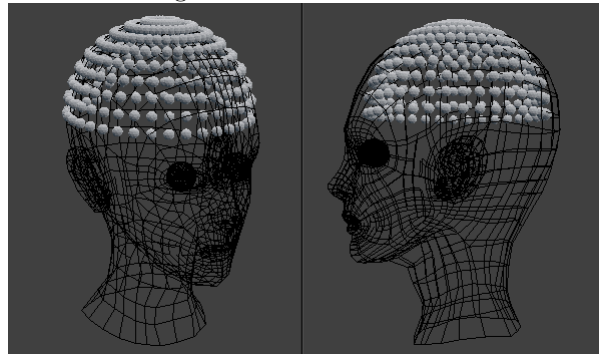
A total of 342 polylines in our generative model is used to describe hair structure. We construct each polyline from 10 points in a 3D space. The resulting feature vector is 10260 dimensions long.

3.3 Approximating Generative Parameters from Input Data

Approximation begins by parsing the geometric data of mesh files into a graph data structure of nodes and edges. The axes of OBJ files are misaligned from the axes 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 interested in 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.
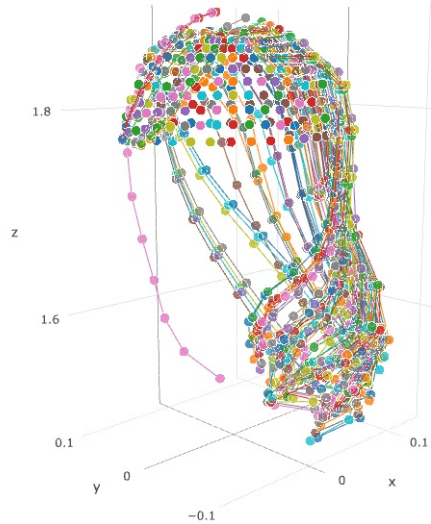
Figure 3.5: A possible configuration of 10260 data points in the generative model

- 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: mesh data file
Output: vertex dictionary of 3D points, mesh graph

---

initialise vertex dictionary, 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

3.3.1 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]:

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.
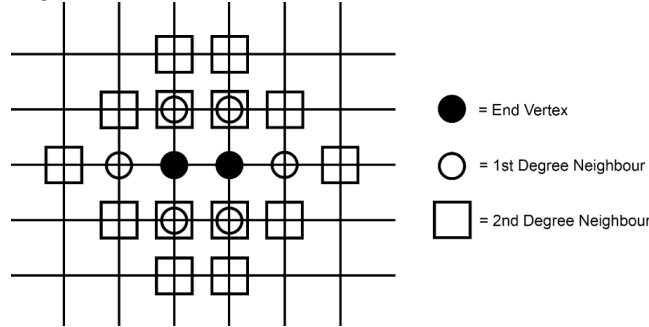
3.3.2   Boundary and Root Edge Loops

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.

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 when the operation is unable to connect any segments further.

Input: vertex dictionary, mesh graph
Output: list of boundary edge loops

---

initialise corner vertices, border vertices, list of boundary edge loops
while *number of corner vertices* **in** *mesh* **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 boundary edge loops collection
            remove vertices of extracted loops from boundary 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 boundary 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.
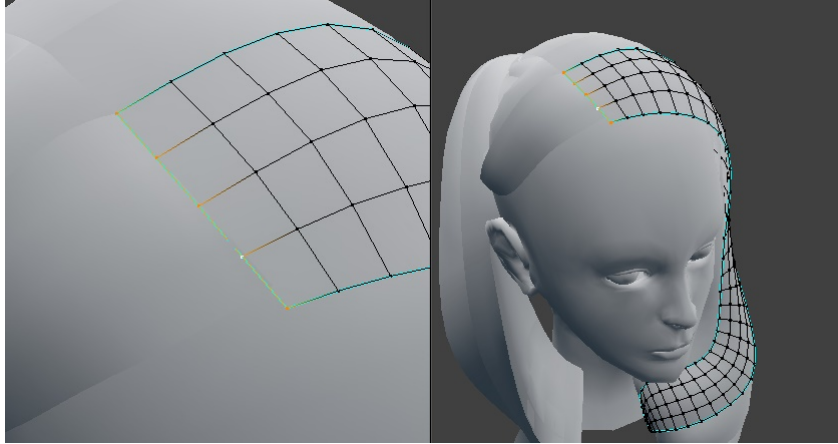
3.3.3 Pivotal Strand Polylines

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 polyline, a sequence of connecting lines.

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

Input: hair mesh
Output: collection of pivotal polylines

---

initialise
for *sub-mesh* ***in*** *mesh* **do**
    find boundary edge loops of sub-mesh
    determine root edge loop among boundary edge loops
    for *node* ***in*** *root edge loop* **do**
        for *neighbour of node* **do**
            if *node* ***not in*** *root loop* **then**
                grow pivotal strand edge loop
                add pivotal strand to polyline collection
            end
        end
    end
end
return pivotal poly-line collection

Algorithm 3.3: Extracting polyline edge loops

3.3.4 Parametrising a Strand Polyline

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

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

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

where s_l is the polyline length.

We determine the indices $[j, j+1]$ where the sample distance lies on the polyline 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 polyline, 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 polyline is to concentrate sampling points at areas of high curvature rather than at evenly spaced intervals. Sampling effectively allows the model to be descriptive with less parameters, mitigating problems introduced by the curse of dimensionality.

3.3.5 Structure Estimation

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

- *proximity* - how close the subject strand is to the root position.

- *significance* - how definitive a particular strand is, or the information conveyed by the subjected polyline.

- *uniqueness* - whether there are other roots that have already captured the information conveyed by the subjected strand.

Several selection operators experimented are:

- *nearest strand selector* - it naively selects the closest strand 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 strand that has maximises the minimum distance from any polyline already selected by other roots.

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 2.9 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.
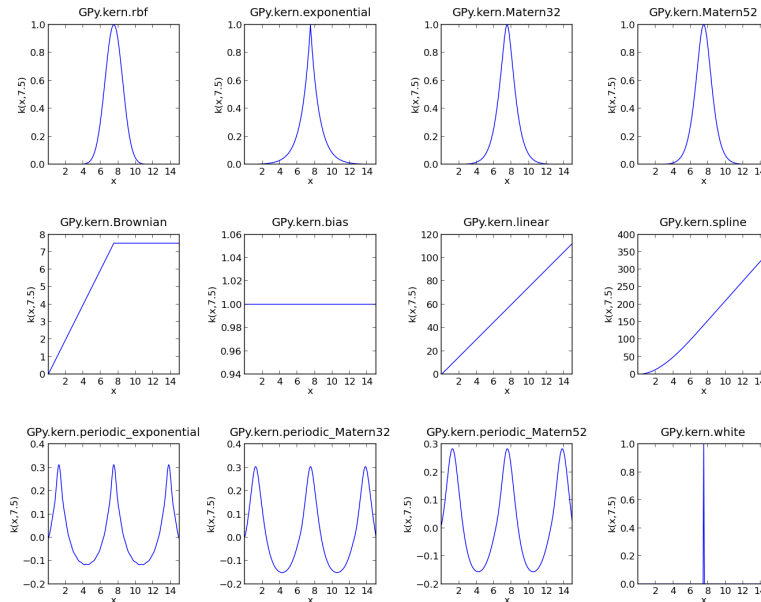
Figure 3.8: Source: GPy Library Documentation: A kernel overview. The GPy library offers various standard kernels. Multi-modal kernels can be formed from combining kernels to represent data of complex nature.

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 3.9 illustrates the results of RBF, exponential, and linear kernels in our regression model.

Figure 3.9: 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 strands* from the polyline data structure of the Blender API. Polygon mesh geometry is produced from assigning these guide strands 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.
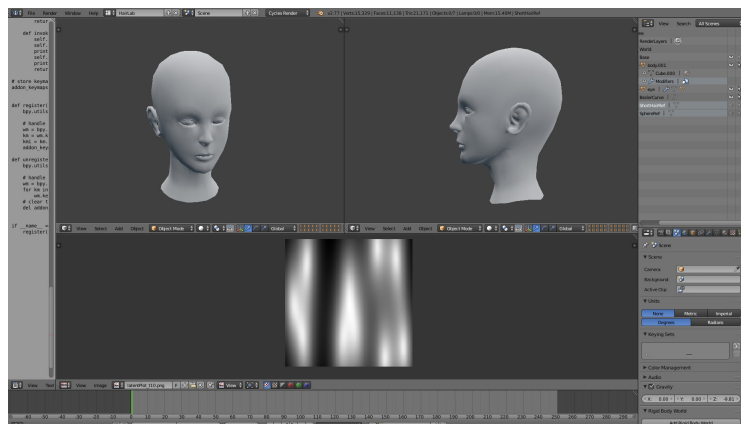
Figure 3.10: The latent hair modelling add-on for Blender 3D uses the image editor to display the latent manifold.

3.6 User Activity Log

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.

Chapter 4

Critical Evaluation

This chapter will start by examining two methods devised to obtain feedback regarding the application of 3D modelling with a latent manifold. First, a survey is used to compare results of the generative output to the training set, and obtain initial reception to the concept. A sample size of 15 subjects completed this survey. Secondly, a sequence of experiments were conducted by users examining our prototype implementation. Activity data was logged to analyse behavioural patterns among subjects. A sample size of 10 completed the experiment. We then proceed to discuss the performance of our demonstrative implementation. The chapter concludes with evaluation of choices made during this project and potential applications of assisted real-time content generation with latent regression models.

4.1 Latent Hair Modelling Survey

The survey starts with a section that queries the name of participants and their experience with producing 3D content. Out of ethical concern, this survey questioned only information necessary for the study, minimising inquiry of personal details. 60% of the participants considered themselves non-experts, while the remaining 40% had some form of experience. The first section seeks to evaluate the performance of our generative model. The survey presents participants a series of images consisting of training meshes and meshes generated using the output of our regression model, to which they give a likelihood rating between the range 0 to 10 for which how much the presented geometry resembles hair.

Figure 4.1: Images presented in the survey, named mesh 1-4 on the first row from left to right, and 5-8 on the second. Training meshes are 1, 4, 5, and 6. Output mesh are 2, 3, 7, and 8.

A mixture of training and output mesh is presented independently without notifying the subjects know which set a particular example belongs to. This enables us to evaluate an objective difference between the perception of training data and output result. If the subject believes that the training meshes are a good indicator for hair, then success criteria for our generative model would be to match those likelihood ratings. Likewise, a low absolute rating for the output mesh is acceptable if the ratings of training meshes are also low.

In general, training mesh is consistently rated very highly in likelihood at 7.92 mean with 2.62 variance.
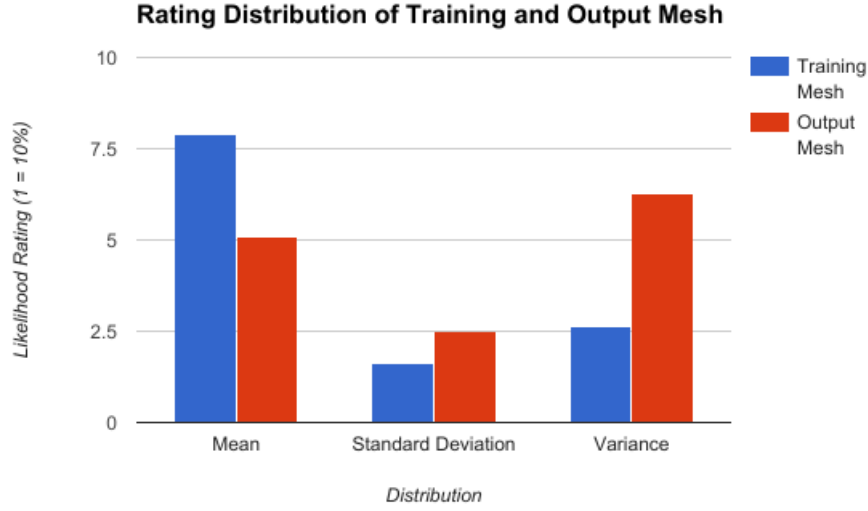
Figure 4.2: The mean, standard deviation, and variance comparison of training and output mesh presented on the survey.

Mesh Type	Mean	Standard Deviation	Variance
Training	7.92	1.62	2.62
Output	5.08	2.51	6.28

Table 4.1: Mean, standard deviation, and variance of training and output mesh (3 s.f).

On the other hand, the output mesh has a mean of 5.08 but results spread out more with a variance of 6.28. To determine whether the difference of results is reliable, we perform an independent samples t-test for inferential evaluation. The null hypothesis is that there exists no statistical significance difference between the samples. The t value is computed as

$$\text{t} = \frac{\text{difference between group means}}{\text{variability of groups}} = \frac{\bar{x_1} - \bar{x_2}}{\sqrt{\frac{\sigma^2}{n_1} + \frac{\sigma^2}{n_2}}}$$

We obtain t-value

$$t = \frac{2.84}{\sqrt{\frac{2.62}{60} + \frac{6.28}{60}}} = 7.37.$$

The degree of freedom for a sample size of 120 is 118. The test rejects the null hypothesis for a p-value of 0.05, where the critical value of a two-tailed test with high degree of freedom is 1.96. This indicates that there is extreme statistical significance between the two sets. From this result, we can infer that there is an observable difference between the training and output meshes.

The following section of the survey introduces basic concepts of 3D modelling and a video depicting the use of a latent manifold for generating meshes. Questions regarding feelings towards use of a latent manifold for 3D production is asked.

When questioned regarding the difficulty of generating output with a latent manifold, the mean is 4.67, suggesting that difficulty is moderate. However, the distribution of responses is skewed, a modal value of 3 tells us that among the sample, most participants found it fairly easy to generate output with a latent manifold.

Calculate quartile percentiles

The idea of using a latent manifold for exploring new hairstyles or variations of existing ones was welcomed. For exploring styles, the mean is 7.53 with a variance of 2.55, whilst exploring variations have

Figure 4.3: Responses for difficulty rating of generating output with a latent manifold, from 1 (easy) to 10 (difficult).
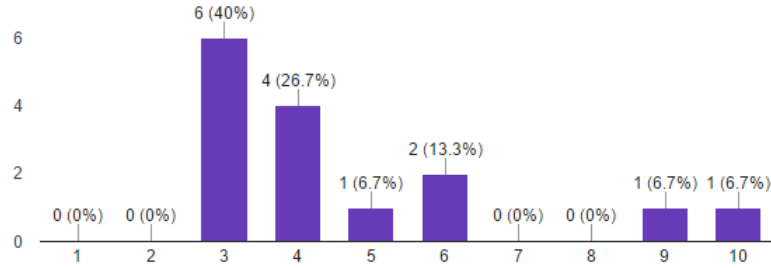
a mean of 7.27 and variance 1.78. In both cases, the high mean value and relatively low variance indicate that participants believe in the potency of discovering new geometry with a latent manifold.

Participants are asked estimate how much of the 3D geometry production pipeline can be automated with a latent manifold. The response resembles a normal distribution, with mean, mode and median at 6, plus variance of 4.

Figure 4.4: Responses for estimation of automating the 3D production pipeline with a latent manifold, from 1 (none) to 10 (all).

An overwhelming portion of participants are willing to try rapid prototyping with a latent manifold. Out of 15 responses, 6 (40.3%) participants agreed that they would use a latent manifold for rapid prototyping, 8 (53.3%) chose 'Maybe', and only one participant decided to not use a latent manifold.
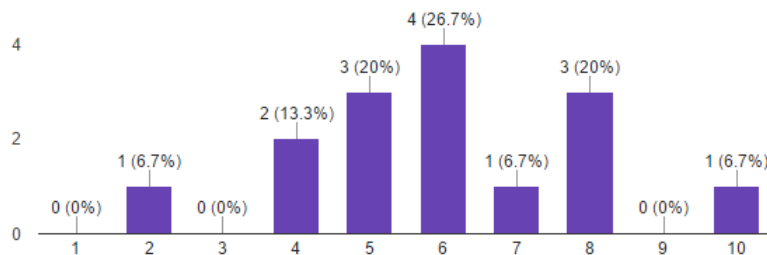
The survey results indicate that whilst the output of our current generative model is notably imperfect when compared to the training data, it is satisfactory expectation as confidence is high in respect to the efficacy of producing 3D content with a latent regression model. Many participants believe a latent manifold is effective for discovering new styles and modifying variations of existing styles. The drawbacks identified include that some users find traversing a non-linear embedded manifold difficult to grasp, thus while it is easy to have an output, controlling the turnout is non-intuitive. That said, most participants would still consider using a latent manifold for rapid prototyping. There is much future research that could yield improved results, to which we will discuss in later sections of this chapter.

Figure 4.5: Responses for whether participants would use a latent manifold for rapid prototyping.
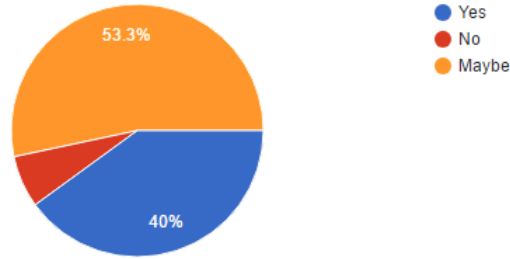
4.2 Latent Application Experiment

Participants are shown a series of images from the output of our regression model, displayed in table 4.2. They are tasked to find the closest corresponding point on the latent manifold that reproduces the geometry presented. An output file with the data displayed on table 4.3 writes the logged user activity obtained from the tests.

Table 4.2: Experiment Geometry
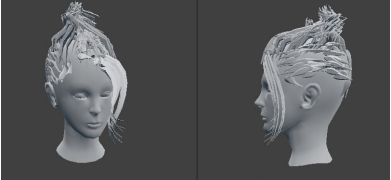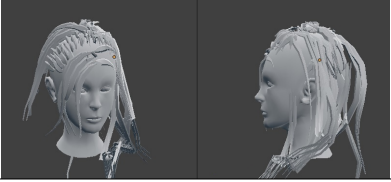
Test Number	Image	Regression Likelihood	Manifold Image Position
1		0.929	12, 24
2		0.925	45, 19
3		0.902	93, 10

Table 4.3: Contents of the output data file.

Data field	Description
Model	The name of the regression model loaded
Time taken	Measure of time spent from starting selection to choosing a result (seconds)
Path	A list of image coordinates sampled by the cursor during latent selection
Distance	The total distance traversed by the the path (image pixels)
Likelihood	The normalised probability associated by the regression model
Latent Variables	The selected latent variables
Generative Parameters	The predicted output of the selected latent variables by the regression model

Behavioural Observation: Familiarising with the manifold

Figure 4.6: Results of a participant who located later targets quickly after spending some time to explore the manifold first.
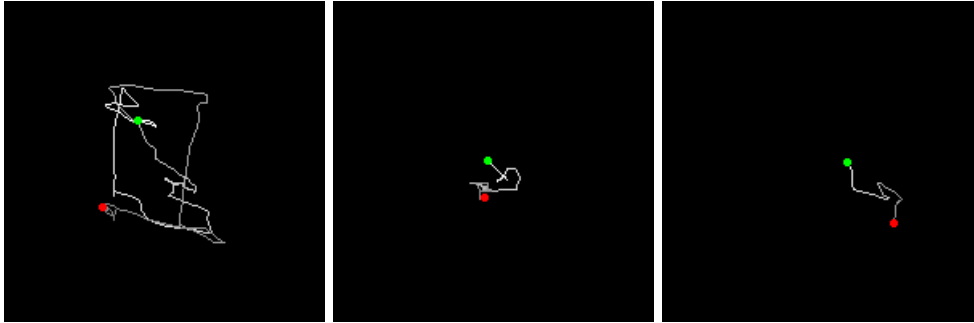
Some users invest in exploring the manifold thoroughly when first introduced to it. One example is displayed on figure 4.6. After familiarising with a particular manifold, participants are able to locate their objective quickly with improved accuracy.

Back up with stats

Behavioural Observation: Fine-tuning the result

Figure 4.7: Plot for the first three tests of a participant.
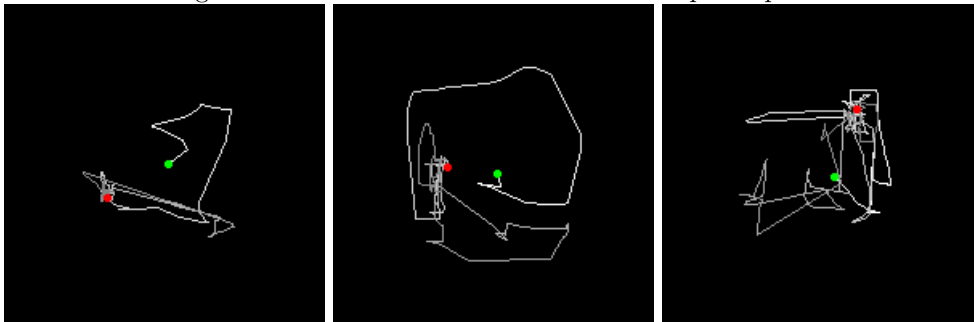
Among participants who take significantly longer to select a prediction, some do so as they value precision over speed. Figure 4.7 shows one such case. We can observe that the participant begins with an exploratory mindset as the path moves boldly, but once they close in to their target, an exploitative strategy is adopted. Staggering movement sets in towards the end of the path as participants attempt to find their best fitting prediction.

Behavioural Observation: Misleading local optima

Participants that do not explore the manifold risk becoming stranded within a local optima. Results in figure 4.8 is an example of this. The participant did not explore during the first test, which lead to a lengthy search process during the second test - where the participant primarily looked within an area that resembles the target, but the actual target is located elsewhere in the manifold. This is a property that makes the latent manifold unintuitive to use, as it would seem logical for similar meshes to be close in the manifold, but a non-linear embedding means that this is not always true.

4.3 Functional Performance

Figure 4.8: Plot for the first three tests of a participant.
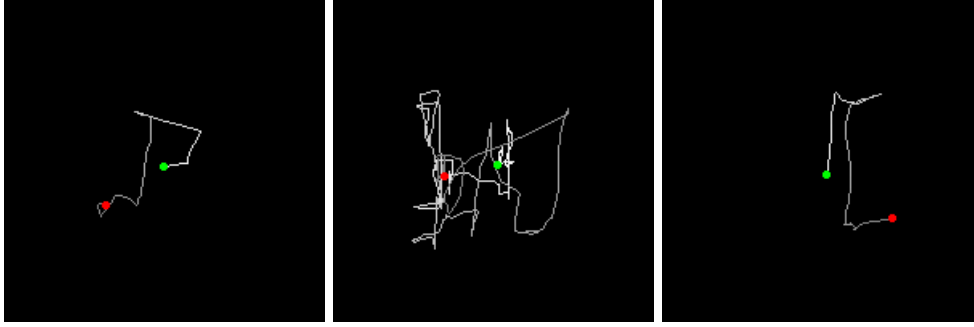
functional testing, including analysis and explanation of failure cases. nice resolution, decent output, measure realtime bald spots, inside head, intersection causes and solutions

A framework that puts our concept of learning-based assisted content generation to use can be divided into two main stages. First, we have the training process which involves:

1. Data acquisition.

2. Kernel and hyper-parameter selection.

3. Learning the regression model.

Once a regression model has been trained, it can then be used for production, the regression process that consists of:

1. Loading the regression model.

2. Selecting a latent variable.

3. Predicting output of selected latent variable.

4. Generating geometry with predicted output.

The ambiguous nature of creative content and mesh data structure makes training a regression model for 3D content difficult. We have demonstrated how algorithmic solutions can automate the task of preparing input mesh for learning. Content creators that have quadrilateral meshes will not require the retopologising stage, which is necessary for our training set as it contains limiting meshes triangulated for rendering. The unpredictable structure of mesh representation requires making some assumptions regarding the input mesh topology. Despite so, our implementation degrades gracefully with the presence of flawed training data. From edge loops, we determine the structure of the surface.

Training mesh data is scarce. Non-experts content creators are more so susceptible to the lack of training meshes. Learning-based assisted content generation by non-linear dimensionality reduction is intended to be robust with small training sets. Table 4.4 displays benchmark results of times taken for the training process on our implementation. The benchmark result is highly dependent on many factors including input data, generative model, implementation, and the hardware of the machine. The point to take away is that training a regression model that makes a prediction from selecting variables in a non-linear latent manifold is a reasonably fast, completing in only matter of minutes.

We use a configuration file to determine properties of our generative model and the fidelity of approximation. In our benchmark tests, we alter the intervals of hair roots to change the number of generative parameters learned by the regression model.

Results of table 4.4 show that approximating generative parameters is the most time-consuming procedure. The time required for approximating is dependent on the complexity of the generative model and input mesh. Thus the benchmark displays a linear relation between total approximation time and

Training Size	Generative Parameters	Approximation time (s)	Optimisation time (s)	Total time (s)
	2700	100	41	141
10	10200	164	49	213
	38850	433	86	519
	2700	177	17	194
20	10200	312	38	350
	38850	877	90	967
	2700	265	29	304
30	10200	507	65	572
	38850	1321	159	1480

Table 4.4: Benchmarks of the training process were performed on a machine with an 2.6 GHz CPU (Intel i7-6700HQ), GTX 970M graphics card, and 8GB RAM.

training size. Hair geometry as one of the most complicated to be represented use more generative parameters than other categories of surfaces. The number of generative parameters can be kept in sustainable numbers by choice of generative model.

Plotting the latent manifold as a 2D image provides a visualisation of latent variables and corresponding likelihood. Large training sets incorporate more example styles, but too many reduce ease of use as exploring the 2D non-linear manifold would interpolate unintuitively. Training sets of this solution will inherently be small in cardinality.

A regression model it can be used to predict generative parameters repeatedly. Serialising regression models trained with small sets will result in distributable file sizes. Content creators have the option to collect and share regression models. Refining a GP-LVM regression model is possible by inferring new observations, updating the posterior distribution. The initial time investment required to train a model is made up for if the regression model can reduce total time spent over the course of its usage.

Loading a small regression model and an image is only required once per start-up of a session, and are negligible with contemporary hardware. A 10200-generative parameter regression model on the same machine that performed the benchmark tests spends around 10^{-3} seconds to predict an output, using the output to generate hair geometry takes an about of 10^{-1} seconds. As a user moves the cursor across the manifold, the implementation presents roughly ten hair meshes per second. This performance is sufficient for producing visible feedback as users traverse the latent manifold.

4.4 Evaluative Discussion

4.4.1 Training and Model

The presence of triangulated faces in training data hinders the accuracy of our regression model. Access to a training set of quadrilateral meshes would improve parameter estimation of our generative model, yielding output that is more representative of training mesh.

The prediction interpolates between styles, drastically differing styles give rise to conflicting substructure of hair geometry. Perhaps it would be more intuitive to use with a training set of varying but similar styles. Campbell and Kautz [9] learn manifolds of fonts for individual characters as well as joint manifolds. A single manifold to edit font characters independently allow for more freedom of control. A consideration would be to use a joint manifold for defining an overall hair structure and then refine layers or sections with single manifolds on top. Multiple manifolds divide the problem at hand, reducing outcomes that seem bizarre due to interpolating of non-linear embeddings at the cost of a more preparation required and an involved training process.

Linear, decision trees, neural nets, GPLVM Supervised, unsupervised, genetic Deep GPs and variants

4.4.2   Comparison with existing solutions

The AutoHair implementation by Chai et al. (2016) [10] delivers hair models that is visually superior in comparison to our demonstrative implementation. AutoHair is a fully automatic procedure, our solution is not intended to replace artists but to assist their creation process. On superior hardware (quad-core Intel i7 CPU, NVIDIA GTX 970 graphics card, and 32GB of RAM), it takes less than a minute for AutoHair to process an image for hair modelling. While this is reasonably fast, our implementation generates geometry at close to real-time. In a production environment, immediate observable feedback is strongly desirable as designs frequently change and creators often want to be directing the progression.

AutoHair used a database of 300 exemplar 3D hair models and a neural network trained on 50,000 portrait images. The significantly larger amount of material required for the training process increases to the time required for training. According to the originating paper, preprocessing of portrait images is outsourced and takes around a minute per image. The 3D hair exemplars take less than ten hours, and training the neural network is about eight hours. A trained implementation of AutoHair can help 3D artists create models, provided that there is an input image available for the desired hairstyle. It can also be used to discover hairstyles if a 3D hairstyle space is pre-generated. It is unlikely that content creators will collect the material required for training their own implementation. Our solution is more flexible in this aspect, as it expects small training sets. This makes it much faster in performance, while not a fully automatic process, it is suitable for assisted content generation and a strong contender for rapid prototyping. Accepting meshes as input to produce output meshes is also more fitting for a production environment than a portrait image. Our solution can easily be used for different genres of 3D models by specifying a generative model for the type of surface or choosing one of the many existing generative models developed.

Example-based hair geometry synthesis by Wang et al. (2009) [24] is a hierarchical synthesis framework that creates hair models by jointly parametrising hair as a 3D vector field and a 2D arrangement of hair strands. Fully automatic without user intervention. 100,000 strands takes less than two minutes on a 3GHz Pentium 4 CPU with 2GB RAM. The output is textured (texture synthesis), completes more of the production pipeline than our solution. Output is controlled by parameters. No learning involved, not flexible. Creates novel hairstyles by taking input hairstyle and find statistically similar spatial arrangements of strands and geometric details. We can use this synthesis framework in place of our generative model.

4.4.3   Potential Improvements

Generative model Regression model Simulating foliage, grass. transferring to fixed objects like a face: mouth, eye, etc that is more "predictable". Determining threshold of acceptable hair. 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.

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.

4.4.4   Applications

The advent of 3D printing, augmented reality, and virtual reality has resurfaced interest in production of 3D representations. In addition of contribution toward the important production pipeline, content creation extends to consumer applications. Advancements in communication and technology established a culture of creation and sharing experiences on social media. An easy to use method of creating content is a coveted feature. For example, consider a system in a video game that allows users to create customise their avatar appearance. A content customisation system desires to be flexible, but may be abused to

create inappropriate content if no restrictions are applied. This is especially problematic if such content is shared or displayed through an online platform. Our solution can be applied to address this problem training a regression model for parameters of the customisation system and only accept inputs that have an associated likelihood above a certain threshold. Selecting from a latent manifold can update many parameters simultaneously and interpolate between predefined configurations, providing an excellent level of customisation while adhering to defined constraints.

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] Blender manual: Edge loop selection. https://wiki.blender.org/index.php/Doc:2.4/Manual/Modeling/Meshes/Selecting/Edges. [Online; accessed 06-May-2017].

[2] S4 cas tools. http://www.modthesims.info/download.php?t=582348. [Online; accessed 06-May-2017].

[3] s4pe. https://github.com/s4ptacle/Sims4Tools. [Online; accessed 06-May-2017].

[4] The sims resource. https://www.thesimsresource.com/. [Online; accessed 06-May-2017].

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

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

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

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

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

[10] 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.

[11] 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.

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

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

[14] H. Hotelling. *Analysis of a Complex of Statistical Variables into Principal Components.* Journal of Educational Psychology, Vol 24(6), 1933.

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

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

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

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

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

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

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

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

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

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

[25] 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.