Novel Approaches for Generating Video Textures

Wentao Fan

A Thesis

in

The Concordia Institute

for

Information Systems Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Applied Science (Information Systems Security) at

Concordia University

Montréal, Québec, Canada

July 2009

Your file Votre référence
ISBN: 978-0-494-63107-2
Our file Notre référence
ISBN: 978-0-494-63107-2

Abstract

Novel Approaches for Generating Video Textures

Wentao Fan

Video texture, a new type of medium, can produce a new video with a continuously varying stream of images from a recorded video. It is synthesized by reordering the input video frames in a way which can be played without any visual discontinuity. However, video texture still experiences few unappealing drawbacks. For instance, video texture techniques can only generate new videos by simply rearranging the order of frames in original videos. Therefore, all the individual frames are the same as before and the result would suffer from "dead-ends" if the current frame could not discover similar frames to make a transition.

In this thesis, we propose several new approaches for synthesizing video textures. These approaches adopt dimensionality reduction and regression techniques to generate video textures. Not only the frames in the resulted video textures are new, but also the "Dead end" problem is avoided. First, we have extended the work of applying principal components analysis (PCA) and autoregressive (AR) process to generate video textures by replacing PCA with five other dimension reduction techniques. Based on our experiments, using these dimensionality reduction techniques has improved the quality of video textures compared with extraction of frame signatures using PCA. The synthesized video textures may contain similar motions as the input video and will never be repeated exactly. All frames synthesized have never appeared before. We also propose a new approach for generating video textures using probabilistic principal components analysis (PPCA) and Gaussian process dynamical model (GPDM). GPDM is a nonparametric model for learning high-dimensional nonlinear dynamical data sets. We adopt PPCA and GPDM on several movie clips to synthesize video textures which contain frames that never appeared before and with similar motions as original videos. Furthermore, we have proposed two ways of generating real-time video textures by applying the incremental Isomap and incremental Spatio-temporal Isomap

(IST-Isomap). Both approaches can produce good real-time video texture results. In particular, IST-Isomap, that we propose, is more suitable for sparse video data (e.g. cartoon).

Acknowledgements

First and foremost, I would like to express my greatest gratitude to my supervisor Dr. Nizar Bouguila. He is such a wonderful advisor, mentor and motivator. I learned a lot from his valuable tutoring, not only technical knowledge but also about dealing in real life. I will be always grateful to him for his support and persistent encouragement.

I would also like to give my appreciation to all professors in CIISE, especially Dr A. Ben Hamza for his patience and guidance through my INSE6510 project.

Many thanks to my colleagues in the lab, for their helpful suggestions during my two years study.

And finally, I would like to thank my family for unconditional support throughout my studies, your endless love and care always encourage me all the time.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Recently, in the computer graphics community, image-based modelling and rendering (IBMR) techniques have drawn considerable attention because of their potential to generate realistic images [1] [2] [3] [4]. In IBMR, a collection of existing images are used to render novel views rather than geometric primitives. IBMR techniques provide a simpler and faster solution for representing more complex scenes in the real world.

Video texture, introduced by Schödl *et al.* [5], has common features with image-based rendering (IBR) technique. It can reuse a recorded video to make a new video stream without visual discontinuity by changing the order of the original frames. Therefore, video texture can be viewed as a "video-based rendering" technique. Video rewrite [6] is the most similar technique to video texture. It can create a new video by using an existing video footage of a speaking person. This technique can also be used in movie production to match new sound track with an actor's lip.

Nowadays, photographs and videos are the most widely used mediums to conduct information. They can be seen everywhere in our daily lives, but each of them has its own drawbacks. A photograph is a static image with a time-invariant information, but without dynamic properties. There are many natural phenomena that cannot be represented by a single snap shot photograph (e.g. leaves swinging in the breeze, water descending from a waterfall, snow falling from the sky). On the other hand, a video is able to capture the dynamic properties of a phenomena, but with limited time.

Video texture is a new type of medium that can be placed somewhere between static images and dynamic videos. It incorporates the properties of a video and a photograph in order to generate

a continuous, infinitely changing stream of images. The term "video texture" is used because it is very similar to image texture. The frames in the original source video are randomly rearranged to generate a new video texture. Although the individual frames are still the same as before, the whole video sequence is infinitely varying. A video texture combines the properties of a photograph and a video to produce a dynamic and timeless result, which can be applied in many different fields. Indeed, a displayed digital photo can be replaced by a video texture. For instance, in the digital photo frame, a video texture of a waterfall with water descending can be used instead of a static picture of waterfall. Or in the cell phone, iPod or computer, you can display a video texture of your child with continuously changing facial expressions like smiling or making funny faces instead of one single expression. Besides, video textures can be used on web pages to replace static images and make the web pages more vivid and interesting. Some other applications related to movies or computer games are also possible.

1.1 The Video Texture Technique

In order to generate a video texture, a video clip or a sequence of still images can be used as an input followed by three steps: analysis, synthesis and rendering. The analysis step is to uncover good transition points by extracting and examining each frame in the original input video clip. The synthesis step synthesizes a new video stream from the analyzed video clip by rearranging the order of the original video frames. The rendering step makes the sequence of video frames together in a way with minimal visual discontinuities. Then a new video texture is created at the end of the generation process.

1.1.1 Analysis

The first step of creating video textures is to analyze the data from the input video. It involve extracting every frame out from the input video and computing the similarities between all pairs

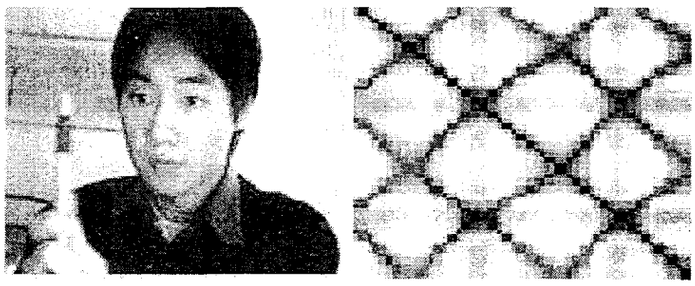


Figure 1.1: An example of the distance matrix (right) from the input video (left).

of them. Before calculating the similarity, it is often required to equalize the brightness in frames in order to reduce the visual discontinuities from one frame to another in the video texture. Additionally, the input video may contain some shakes because of bad capturing conditions. Thus, it is necessary to apply a video stabilization tool to make the video clip more stable.

According to the original algorithm of video texture [5], L_2 distance is used to compute the dissimilarity between frames with the result stored in the matrix

$$D_{ij} = \|I_i - I_j\| \tag{1}$$

here, D_{ij} is the matrix that represents the L_2 distance between each pair of images I_i and I_j. Figure 1.1 gives an example of distance matrix (right) from an input video (left). In this video, a person moves a pen continuously in front of the camera. Since video texture reorders the original video frames into a new sequence, it is important to insure that the new transitions are visually smooth. More precisely, a transition from frame i to frame j occurred when the successor of frame i in the original video is substituted by the frame j and the predecessor of frame j is substituted by the frame i. Thus, frame i should be similar to frame $j-1$ and frame $i+1$ should be similar to frame j. This also means that the value of $D_{i+1,j}$ needs to be small if we want to make a transition from frame i to j without visual discontinuity.

Basically, video textures are Markov processes, with each video frame represents a state. The Markov process can be represented as a matrix of probabilities, and each element P_{ij} in the matrix

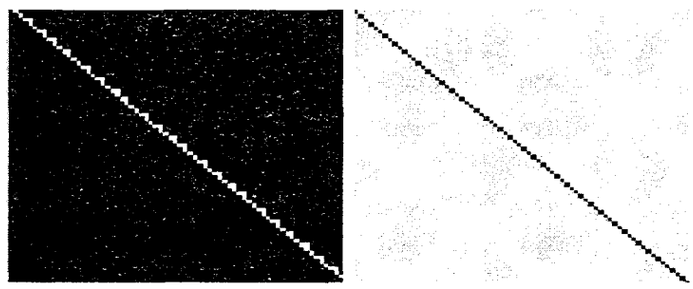


Figure 1.2: Example of the probability matrix with different parameter σ, $\sigma = 1$ on the left and $\sigma = 10$ on the right.

is the probability of a transition from frame i to frame j. The frame-to-frame distance D_{ij} can be mapped to the probabilities through an exponential function

$$P_{ij} \propto \exp(\frac{-D_{i+1,j}}{\sigma}) \tag{2}$$

here, the parameter σ controls the mapping between the L_2 distance and the probability of transition. Smaller values of σ means only to select the very best transitions and larger values of σ allow for greater variety and relatively poorer transitions. Figure 1.2 gives an example of probability matrix with $\sigma = 1$ and $\sigma = 10$, respectively. All the probabilities for the transitions from the same frame should be normalized so that $\sum_j P_{ij} = 1$. The transition from a frame to the next is done according to the distribution of P_{ij}.

Preserving dynamics

The similarity is not the only factor needed to be considered for a good transition, but also the dynamics of the system is needed to be preserved. Figure 1.3 also shows that there are two possible transitions from frame i to either frame j_1 or j_2, but only j_2 has the same dynamics as i, which is swinging from left to right. On the other hand, if the transition is from frame i to frame j_1, it will result in sudden and unpleasing directional changes for the moving pen.

Thus, it is necessary to enforce the coherence in the direction of the movement. One possible

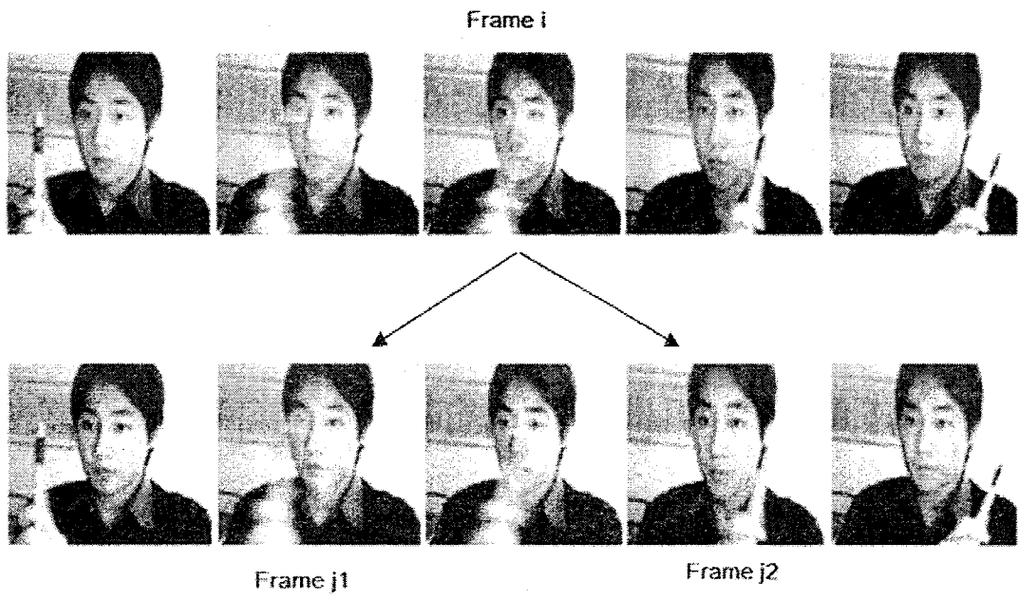


Figure 1.3: Example of preserving dynamics

solution is to consider the similarities of the neighboring frames as well rather than just the current frame. This subsequence match can be accomplished by filtering the L_2 distance matrix $D_{i,j}$ with a diagonal kernel with weights $[w_{-m}, ..., w_{m-1}]$

$$D'_{ij} = \sum_{k=-m}^{m-1} w_k D_{i+k,j+k} \tag{3}$$

here, the parameter m controls the size of the window. In the implementation, m is set to 1 or 2 to make it a 2 or 4 tap filter with binomial weights. And the new probabilities P'_{ij} after filtering can be given as

$$P'_{ij} \propto \exp(\frac{-D'_{i-1,j}}{\sigma}) \, . \tag{4}$$

After the filtering, the unacceptable transitions will be eliminated.

Avoiding dead ends

The algorithm so far can create good results, but it only considers the local cost for the given transition. This means that only the similarities and dynamics between two frames are involved in a transition; there is no consideration for the situation in which the transition may lead a video into a "dead end". A "dead end" problem happens when a transition is made from one frame to another and this will lead into some portion of the video which there is no graceful exist, the video will be stuck forever.

In order to solve this problem, the anticipated future cost for choosing a transition needs to be considered. Let $D''_{i,j}$ be the future cost of a transition from frame $i-1$ to frame j, this cost represents the expected average cost of future transitions. $D''_{i,j}$ is defined by summing up all future anticipated costs and k is the number of future transitions

$$D''_{i,j} = (D'_{i,j})^p + \alpha \sum_k P''_{jk} D''_{jk} \tag{5}$$

here, p is a constant used to control the tradeoff between taking multiple good transitions from frame i to j or just a single poor transition. Normally (but not always) p is set to an integer between 1 and 5 (you can choose between a single transition or up to 5 transitions). The parameter α is the convergent rate used to control the relatively weight of the future transition in the metric. For convergence, α must be set as $0 < \alpha < 1$; in practice, it is set as $0.99 < \alpha < 0.999$. P''_{jk} is similar as in (2)

$$P''_{ij} \propto \exp(\frac{-D''_{i+1,j}}{\sigma}) \, . \tag{6}$$

(5) and (6) can be solved using iterative algorithm, but it is very slow to converge. To improve the algorithm, since as $\sigma \rightarrow 0$, the value of probability P''_{jk} in (6) will be close to 1 for the best transition. (5) can be modified as

$$D''_{i,j} = (D'_{i,j})^p + \alpha \min_k D''_{jk} \, . \tag{7}$$

The above equation is known as Q-learning [7] in the reinforcement learning community, and it

has been proven to always converge. D''_{ij} is initialized with $D''_{ij} = D'_{ij}$ and the value of $D''_i j$ is obtained by iterating until it becomes stable.

1.1.2 Synthesis

After good transition points are found, the next step is to decide the order of playing video frames from the input video. There are two methods to sequence the frames into a video texture: random play and video loops.

Random play

Random play is the simplest and fastest method to create a video texture. Basically, the video texture can be started at any frame which has the transition probability larger than zero. The next frame is selected by walking through all other frames and choosing one with higher transition probability. A new video texture can be generated by repeating the above steps. This generation algorithm is very simple and fast, but it produces very poor results which contain many visible jumps and discontinuities.

Video loops

Video loops are needed when playing video textures using existing digital video players or web browsers. For the video textures produced by finite duration video loops, they do repeat in a fixed period and can be continuously played in a standard "loop mode" in the player without any visible discontinuities.

Video loops is a more difficult and complex synthesis method but also produces better results. The general idea is to find out transitions in the original video that have similar beginning and end. Then it may be possible to jump from the end of this transition to its beginning without any visible jumps. This kind of transition is known as a primitive loop. One or more primitive loops will be

combined into cyclic sequences which are called compound loops. Using such compound loops to jump from one to another will give video texture a much more smooth and natural look.

1.1.3 Rendering

Even though there is only a small amount of discontinuities in the sequence after the analysis and synthesis steps, the transitions produced by jumping from one frame to another are still noticeable in the video texture. Therefore, the cross-fading algorithm is applied in the rendering step. By using this method, frames of the sequence before and after the transition are blended together, which means that the frames around the source of the transition are linearly faded out, while the ones around the destination of the transition are faded in. In this implementation, a cross-fading algorithm is provided as

$$B(x,y) = \sum_i \alpha_i I_i(x,y) \tag{8}$$

here, α_i is the blending weight and is normalized such that $\Sigma_i \alpha_i = 1$.

After the rendering step, a continuously, infinitely varying video texture is generated with a very smooth look and much less visible discontinuities.

1.2 Video Sprites

Video sprite is an extension of the video texture [8] [9]. The term "video sprites" was introduced by Pollard *et al.* [10], the idea is also used in Finkelstein *et al.*'s multi-resolution video work [11]. Instead of using the whole image to generate video textures, we can just use an object in the image. First this object can be shot against a green or blue screen and then use background-subtraction method to extract the object from the background, which is considered as a sprite. Similar to video texture algorithm, new video sprite animations can be made by replacing all the sprites in an appropriate visually appealing order. The new animation of the object can be placed at an arbitrary image location and controlled by defining a cost function [12]. This means that the desired

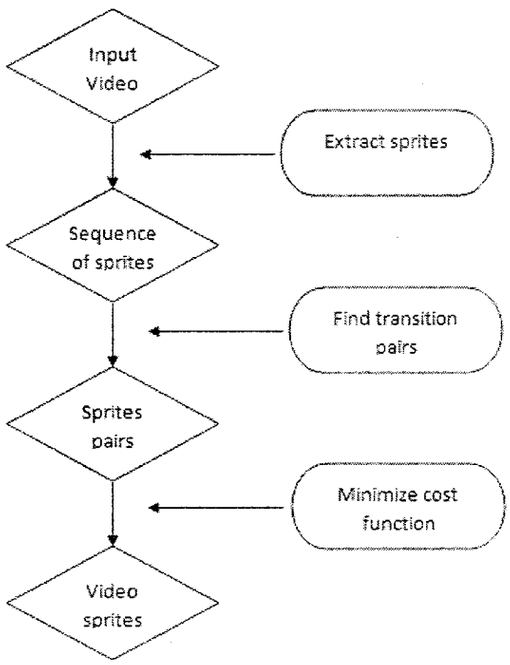


Figure 1.4: The process of generating video sprites

animation can be archived by optimizing the parameters of the animation which in turn minimizes the cost function. Figure 1.4 shows the process of generating video sprites.

1.3 Extensions and Research Works Based on Video Texture

The idea of video texture has inspired many other researchers. Li and Shum [13] presented a new technique called motion texture which is a two-level statistical model that can use the existing motion capture data to synthesize complex human-figure motions (e.g. dancing). Celly and Zordan [14] introduced a technique based on pre-recorded footage, which can control characters in video-based animation in order to generate new realistic sequences. Some other similar works [15], [16]

and [17] also provide solutions for finding good transitions between motion capture data samples in order to create new and longer continuous sequences of motion data. These video-based rendering works can be used to create background elements and special effects in games or movies.

Additionally, there are many attempts on the application of new algorithms to create video textures. For example, Campbell *et al.* [18] employed an auto-regressive process to generate video textures. First, each frame from the input video is transformed into an eigenspace using Principal Components Analysis (PCA), which in turn allow the original sequence to be viewed as a signature through this low-dimensional space. Then a new sequence is created by moving through this space and creating similar signatures.

Another related work introduced by Kalnins *et al.* [19] is quite interesting. They created a system to let artists and designers directly annotate 3D models with strokes. In other words, artists can choose the "brush" style to draw strokes and convey the aesthetic style onto a 3D object from different view angles. Part of Kalnins's work closely followed the algorithm of video textures. They synthesized new sequences of stroke offsets from the set of example strokes rather than synthesized new sequences of frames from an existing video footage. Vivek *et al.* [20] introduced a new algorithm for image and video texture synthesis by using graph cuts which can improve the quality of video textures. Briefly, they synthesized a new texture by copying and transforming patches from sample images or video to the output and then stitch them together along optimal seams.

One of the most interesting extensions for video texture is panoramic video texture (PVT) introduced by Agarwala *et al.* [21]. A series of photos were taken from a single view point and then stitched into a single large image, which is called panoramic image. Panoramic video has the same concept as panoramic image but only needs to replace photos with videos. Nowadays, many web sites have panoramic images or videos, especially web pages for hotels or tourism places. Those images can provide people a much more immersed sense than just a single snapshot. Also, it can be used as a forensic tool to reconstruct criminal scene.

1.4 Contributions

The contributions of this thesis are as follows:

☞ **Compare Different Dimensionality Reduction Techniques for Creating Video Textures:** Recently, a new method of generating video textures has been proposed. It first applies principal components analysis (PCA) to extract signatures or patterns from the original video sequence; and then implements an autoregressive process (AR) model to synthesize new video textures. In this thesis, we extend this video texture generation method by comparing PCA with other dimensionality reduction techniques such as probabilistic principal components analysis, kernel principal components analysis, independent component analysis, local linear embedding and Isomap. According to our experiments, these approaches outperform the original approach by providing us video textures with better quality.

☞ **Generate Video Textures by PPCA and Gaussian Process Dynamical Model:** We propose a new method of generating video textures by implementing probabilistic principal components analysis (PPCA) and Gaussian Process Dynamical model (GPDM). Compared to the original video texture technique, video texture synthesized by PPCA and GPDM has the following advantages: it is able to generate new video frames that have never existed in the input original video clip; the "dead-end" problem is totally avoided; and it could also provide video textures that are more robust to noise.

☞ **Generate Real-time Video Textures by Incremental Isomap and Incremental ST-Isomap:** We also propose two different approaches for synthesizing real-time video textures. The first approach is applicable to generate real-time video textures using incremental Isomap and Autoregressive (AR) process. Nonetheless, it does not extend well for videos that are more sparse, such as cartoons. Inspired by spatio-temporal Isomap (ST-Isomap), we propose an extension of incremental Isomap, which contains spatio-temporal coherence in the data set and can also be applied in an incremental mode. We name it as Incremental spatio- temporal

Isomap (IST-Isomap). Our second real-time video texture generation approach exploits IST-Isomap and AR process. It provides more efficient and better quality results for sparse videos than the first approach. Compared with other video texture generation approaches, both of our approaches are able to synthesize new video textures in a real-time fashion which in turn offer advantages (e.g. faster and more efficient) in applications where data are sequentially obtained.

1.5 Thesis Overview

The organization of this thesis is as follows:

- Chapter 1 introduced the original video textures technique. It provides the background information for the following chapters.
- In Chapter 2, we extend a recently proposed video texture generation method (based on PCA and AR process) by comparing PCA with five other dimensionality reduction techniques. The comparative results and case studies demonstrate that these approaches prevail the original approach by providing us video textures with better quality.
- In Chapter 3, we propose a new method for generating video textures by applying PPCA and Gaussian GPDM. The experimental results demonstrate improved performances according to the robustness to noise compared with the standard video textures technique.
- In Chapter 4, we propose two different approaches for synthesizing real-time video textures. The first approach applies incremental Isomap and AR process, which works well except for sparse videos. The other one applies incremental ST-Isomap and AR process, which provides better results for sparse videos (e.g. cartoon) than the first approach. Our experimental results show that these two real-time approaches can process sequential data more efficiently than the batch approach.

- In Conclusions, we summarize our contributions.

Chapter 2

Generate Video Textures by Dimensionality Reduction Techniques

Video texture is a new type of medium which can provide a new video with a continuously varying stream of images from a recorded video. It is created by reordering the input video frames in a way that can be played without any visual discontinuity. Recently, a new method of generating video textures has been proposed. It first applies principal components analysis (PCA) to extract signatures or patterns from the original video sequence, and then implements an autoregressive process (AR) model to synthesize new video textures. In this chapter, we extend this video texture generation method by comparing PCA with other dimensionality reduction techniques such as probabilistic principal components analysis, kernel principal components analysis, independent component analysis, local linear embedding and Isomap. According to our experiments, these approaches ourperform the original approach by providing us with better quality video textures.

2.1 Introduction

Video texture which has been introduced by Schödl *et al.* [5], is a new type of medium that can generate a continuous, infinitely changing stream of images from a recorded video. The term "video texture" is used because it is very similar to image textures. This technique can be considered as video-based rendering (VBR) because it has similar features with image-based rendering (IBR) technique [1], that is, both of them are able to reuse the already existing resources to synthesize

new objects. For video textures, a recorded video is used to make a new video stream without any visual discontinuity by changing the order of the original frames. This would be useful in movie and game industries, since it may create new objects by reusing existing resources so that time and human resources may be saved. More applications may be found in [9] [8] [21] [13]. However, same as the original video textures technique, all of these works can only generate new videos by just switching the order of frames and the results would suffer from 'dead-ends'.

Recently in computer vision, there are increased number of researches on time series analysis to model the dynamical characteristics of complex systems. Autoregressive (AR) process [22] is a tool used for understanding and predicting future values in a time series. In [23], Fitzgibbon have introduced a new method for creating video textures by applying principal components analysis (PCA) and AR process. All frames in the generated video are new and consist with the motions in the original video, and 'dead ends' would never appear. In [18], Campbell *et al.* have extended this approach to work with strongly non-linear sequences by applying a spline and a combined appearance model.

The rest of this chapter is organized as follows: We briefly review different dimensionality reduction techniques in section 2.2. Then, we compare the experimental results by applying different dimensionality reduction techniques to generate video textures in section 2.3.

2.2 Dimensionality Reduction Approaches

Dimensionality reduction is an important research topic in the area of data analysis. The goal of dimensionality reduction techniques is to discover a low-dimensional subspace that best represents a given set of data points. In this chapter, we extend the work of Fitzgibbon [23] by comparing PCA with other representative dimensionality reduction techniques to extract signatures from video frames and then synthesize new video textures. The techniques we have applied are: probabilistic principal components analysis, kernel principal components analysis, Isomap, local linear embedding and independent component analysis. In this section, we will give a brief introduction

for each technique individually.

2.2.1 Probabilistic principal component analysis

PCA is a quite effective technique for dimensionality reduction and has been used in different image processing and pattern recognition applications. It is based on a linear projection of input data into a lower dimensional space than the original data space. But one disadvantage of PCA is that there is no associated probabilistic model for the observed data. In [24], Bishop *et al.* propose a probabilistic model for PCA by showing that PCA is able to be represented as the maximum likelihood solution of a probabilistic latent variable model. This novel form of PCA is known as probabilistic PCA (PPCA) [1], and it has some practical advantages compared with PCA:

- PPCA represents a constrained form of the Gaussian distribution in which the number of free parameters can be restricted while still catch the dominant correlations in a data set.
- In PPCA, expectation-maximization (EM) algorithm can be used, this can be computationally efficient by avoiding the computation of the data covariance matrix as an intermediate step.
- By using PPCA and EM algorithm, we can deal with missing values in a data set.
- In PPCA, the existence of likelihood function makes it possible to compare with other probabilistic density models directly.
- PPCA can be used for dimensionality reduction as well as a general Gaussian density model.

[1] We have also applied mixtures of probabilistic principal component analysis [25] to generate the signatures of video frames, but the only good result we obtained is when the mixture size is equal to one which is same as the standard PPCA.

Maximum likelihood estimation for Probabilistic PCA

PPCA can be formulated by first choosing an explicit latent variable $\mathbf{z}$ corresponding to the principal component subspace and then sampling the observed variable $\mathbf{x}$ conditioned on this latent variable. We define a Gaussian prior distribution $p(\mathbf{z})$ for the latent variable, and a conditional Gaussian distribution $p(\mathbf{x}|\mathbf{z})$ for the observed variable $\mathbf{x}$ conditioned on the latent variable $\mathbf{z}$. The latent variable distribution over $\mathbf{z}$ is defined as

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I}) \ . \tag{1}$$

The distribution of $\mathbf{x}$ conditioned on $\mathbf{z}$ is again Gaussian

$$p(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}|\mathbf{W}\mathbf{z} + \boldsymbol{\mu}, \sigma^2\mathbf{I}) \tag{2}$$

where the $D \times M$ matrix $\mathbf{W}$ and the D-dimensional vector $\boldsymbol{\mu}$ are used to regulate the distribution of $\mathbf{z}$. The variance of the conditional distribution is governed by the scalar σ^2.

The observed variable $\mathbf{x}$ can be defined by a linear transformation of latent variable $\mathbf{z}$ and an additive Gaussian noise

$$\mathbf{x} = \mathbf{W}\mathbf{z} + \boldsymbol{\mu} + \epsilon \tag{3}$$

where $\mathbf{z}$ is an M-dimensional Gaussian latent variable, and ϵ is a D-dimensional Gaussian noise with zero-mean and covariance $\sigma^2\mathbf{I}$.

Then, we require to determine the values of the unknown parameters $\mathbf{W}$, μ and σ^2 by using maximum likelihood algorithm. The marginal distribution of observed variable $\mathbf{x}$ can be acquired by

$$p(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z} \ . \tag{4}$$

From (1) and (2), this marginal distribution $p(\mathbf{x})$ is again Gaussian, and is given by

$$p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\mu, \mathbf{C}) \tag{5}$$

where the $D \times D$ covariance matrix $\mathbf{C}$ is defined by

$$\mathbf{C} = \mathbf{W}\mathbf{W}^T + \sigma^2\mathbf{I}\,. \tag{6}$$

Given a data set $\mathbf{X} = \{\mathbf{x}_n\}$ of observed data points, we may obtain the corresponding log likelihood function from (5) as

$$\ln p(\mathbf{X}|\mathbf{W}, \mu, \sigma^2) = -\frac{N}{2}\{D\ln(2\pi) + \ln|\mathbf{C}| + tr(\mathbf{C}^{-1}\mathbf{S})\} \tag{7}$$

where $\mathbf{S}$ is the covariance matrix of the data set and is given by

$$\mathbf{S} = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}_n - \bar{\mathbf{x}})(\mathbf{x}_n - \bar{\mathbf{x}})^T\,. \tag{8}$$

Setting the derivative of the log likelihood to zero with respect to $\mathbf{W}$, we may acquire the solution in an exact closed-form

$$\mathbf{W}_{ML} = \mathbf{U}_M(\mathbf{L}_M - \sigma^2\mathbf{I})^{1/2}\mathbf{R} \tag{9}$$

where $\mathbf{U}_M$ is a $D \times M$ matrix whose columns are composed of the principal eigenvectors of $\mathbf{S}$, and the $M \times M$ diagonal matrix $\mathbf{L}_M$ contains the corresponding eigenvalues $\lambda_1, ..., \lambda_M$. $\mathbf{R}$ is an arbitrary $M \times M$ orthogonal matrix.

Similarly, the maximum likelihood solution for σ^2 is given by

$$\sigma^2_{ML} = \frac{1}{D-M}\sum_{i=M+1}^{D}\lambda_i \tag{10}$$

where σ^2_{ML} is the average variance for the discarded dimensions.

2.2.2 Kernel principal component analysis

Kernel principal component analysis (KPCA) [26], is an extension of the conventional principal component analysis (PCA) using the technique of kernel methods. The operations of standard PCA are linear. Using nonlinear kernels, it is possible to generate nonlinear distributions.

At the beginning, the distribution of the input data is non-linear. One way to deal with such a distribution is to attempt to linearise it by non-linearly mapping the data from the input space to a new feature space. Consider a set of observations $\{\mathbf{x}_n\}$ with dimensionality D, where $n = 1, ..., N$. Instead of implementing standard PCA directly, each data point in $\{\mathbf{x}_n\}$ is transformed into an higher dimensional feature space with dimensionality M by using a nonlinear transformation function $\phi(\mathbf{x})$

$$\mathbf{x}_n \rightarrow \phi(\mathbf{x}_n) \ . \tag{11}$$

Then we can perform standard PCA in this new feature spac. We also assume that $\phi(\mathbf{x}_n)$ has zero mean. We can then calculate the covariance matrix $\mathbf{C}$ by

$$\mathbf{C} = \frac{1}{N} \sum_{n=1}^{N} \phi(\mathbf{x}_n)\phi(\mathbf{x}_n)^T \ . \tag{12}$$

Then the formula of its eigenvector is defined as

$$\mathbf{C}\mathbf{v}_i = \lambda_i \mathbf{v}_i \ . \tag{13}$$

where $i = 1, ..., M$. According to the definition of $\mathbf{C}$, $\mathbf{C}\mathbf{v}_i$ is a linear combination of vectors $\phi(x_n)$. Therefore, we can obtain $\mathbf{v}_i$ by

$$\mathbf{v}_i = \sum_{n=1}^{N} a_{in}\phi(\mathbf{x}_n) \tag{14}$$

where a_{in} is the coefficient of linear combination. Substituting (12) and (14) into (13), we have

$$\frac{1}{N} \sum_{n=1}^{N} \phi(\mathbf{x}_n)\phi(\mathbf{x}_n)^T \sum_{m=1}^{N} a_{im}\phi(\mathbf{x}_m) = \lambda_i \sum_{n=1}^{N} a_{in}\phi(\mathbf{x}_n) \ . \tag{15}$$

Instead of using the scalar product of $\phi(\mathbf{x}_n)^T\phi(\mathbf{x}_m)$, we define a kernel function $k(\mathbf{x}_n, \mathbf{x}_m) = \phi(\mathbf{x}_n)^T\phi(\mathbf{x}_m)$, and multiply both sides of (15) by $\phi(\mathbf{x}_q)^T$, we then have

$$\frac{1}{N}\sum_{n=1}^{N} k(\mathbf{x}_q, \mathbf{x}_n)\sum_{m=1}^{m} a_{im}k(\mathbf{x}_n, \mathbf{x}_m) = \lambda_i \sum_{n=1}^{N} a_{in}k(\mathbf{x}_q, \mathbf{x}_n) \ . \tag{16}$$

By defining a matrix $\mathbf{K}$ with elements $\mathbf{K}_{m,n} = k(\mathbf{x}_m, \mathbf{x}_n)$,the above equation can also be represented as

$$N\lambda_i \mathbf{K}\mathbf{a}_i = \mathbf{K}^2\mathbf{a}_i \tag{17}$$

where $\mathbf{a}_i$ is an N-dimensional vector which contains the elements a_{ni} for $n = 1, ..., N$. By removing a factor $\mathbf{K}$ from both sides of (17), this is equivalent to

$$N\lambda_i \mathbf{a}_i = \mathbf{K}\mathbf{a}_i \ . \tag{18}$$

We can obtain the vector $\mathbf{a}_i$ by extracting the eigenvectors of $\mathbf{K}$. It is also required that the principal component $\mathbf{v}_i$ needs to be normalized. Then, we obtain

$$\mathbf{v}_i^T\mathbf{v}_i = \sum_{n=1}^{N}\sum_{m=1}^{N} a_{in}a_{im}\phi(\mathbf{x}_n)^T\phi(\mathbf{x}_m) = \mathbf{a}_i^T\mathbf{K}\mathbf{a}_i = \lambda_i N\mathbf{a}_i^T\mathbf{a}_i = 1 \ . \tag{19}$$

After solving the eigenvector problem, the principal component projections can then also be expressed in terms of kernel function. A data point $\mathbf{x}$ can be projected onto eigenvector i by

$$f_i(\mathbf{x}) = \phi(\mathbf{x})^T\mathbf{v}_i = \sum_{n=1}^{N} a_{in}k(\mathbf{x}, \mathbf{x}_n) \ . \tag{20}$$

Then we can use this to extract a data point's features when applying kernel PCA.

2.2.3 Isomap and ST-Isomap

Isomap [27] is global nonlinear dimensionality reduction technique which extend multidimensional scaling (MDS) technique. In MDS, we first calculate the pairwise dissimilarity (or similarity) between all pairs of data points, then we try to discover a low-dimensional embedding of a set

of high-dimensional data points. Normally, Euclidean distance is used to calculate the dissimilarity between pairs of data points in MDS. In Isomap, instead of using Euclidean distance, Geodesic distance [28] [29] on the manifold is used as the measurement of dissimilarity.

Isomap algorithm contains three major steps. The first step is to determine the neighborhood relations for all data points by constructing a weighted graph, it uses edge weights to indicate distances between all data points in original space. The second is to discover the geodesic distances between all pairs of data points on the manifold by computing the shortest path distances in the weighted graph. The last step is to apply classical MDS to the dissimilarity matrix formed by the shortest path distances, constructing an embedding of the data points in a low-dimensional space.

Isomap is an efficient technique that is capable to discover a low-dimensional manifold embedded in a high-dimensional space. However, Isomap is not able to uncover the underlying spatio-temporal structure of the data set. Since we are dealing with videos. There exist temporal dependencies between sequentially adjacent frames. This drawback can be solved by the modified version of Isomap: spatio-temporal Isomap (ST-Isomap) [30]. It augments the general Isomap framework to consider the temporal relationships in local neighborhoods of data points. Similar as the standard Isomap, ST-Isomap preserves the intrinsic geometry of the data, and it retains the structure of temporal coherence as well.

More details of Isomap and ST-Isomap are introduced in chapter 4.

2.2.4 Locally linear embedding

Locally linear embedding (LLE) [31] is a local nonlinear dimensionality reduction technique. It is able to compute a low-dimensional embedding of the high dimensional data while preserving the neighborhood structure of the original space. It has been proved that this method guarantees the generation of globally optimal solution and does not involve local minimum.

The LLE algorithm is based on simple geometric intuitions. That is, for a high-dimensional manifold, it can be decomposed into many small patches. If each patch is small enough, it can

be approximated as linear patch. Suppose a data consist of N real valued vectors x_i with dimensionality D, sampled from one smooth underlying manifold. If there exists sufficient data points, we may assume each data point and its neighbors lie on or close to a locally linear patch of the manifold. In the simplest case of LLE algorithm, the neighbors for each data can be calculated by K nearest neighbors in Euclidean distance. Thus, each data point can be represented by a linear combination of its neighbors

$$\mathbf{x}_i \approx \sum_j w_{ij}\mathbf{x}_j \tag{21}$$

here, w_{ij} is the coefficient of the local geometry of each patch. The matrix $\mathbf{W}$ of the linear coefficients for all data points can be computed by minimizing the reconstruction error as following

$$E(W) = \sum_i \parallel \mathbf{x}_i - \sum_j w_{ij}\mathbf{x}_j \parallel^2 \ . \tag{22}$$

There are two constraints that need to be considered for minimizing the reconstruction error $E(\mathbf{W})$: first, each data point $\mathbf{x}_i$ is reconstructed only from its neighbors, $w_{ij} = 0$ if $\mathbf{x}_j$ is not a neighbor of $\mathbf{x}_i$. Second, the rows of the coefficient matrix $\mathbf{W}$ must sum to one: $\sum_j w_{ij} = 1$. The optimal weights $\mathbf{W}_{ij}$ can be obtained by solving a least squares problem.

After finding the weight matrix $\mathbf{W}$, the next job is to obtain a linear mapping of each neighborhood structure from high-dimensional coordinates to a lower-dimensional space. Since the weights w_{ij} reconstructed the data point $\mathbf{x}_i$ in the original space, it is expected that the same weights reconstruct each data point from its neighbors in the lower-dimensional space. Therefore, a mapping matrix $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_N]$ that preserves the local neighborhood information can be obtained by minimizing the embedding cost function

$$\theta(\mathbf{Y}) = \sum_i \parallel \mathbf{y}_i - \sum_j w_{ij}\mathbf{y}_j \parallel^2 \tag{23}$$

here, the vector $\mathbf{y}_i$ is the global internal coordinates of the data point $\mathbf{x}_i$ on the manifold. This cost function is also based on locally linear reconstruction errors, but here we fix the weights w_{ij} while optimizing the mapping matrix $\mathbf{Y}$.

In summary, the LLE algorithm contains three major steps:

1. Compute the K nearest neighbors for each data point $\mathbf{x}_i$.

2. Compute the weights w_{ij} by minimizing the reconstruction error in (22).

3. Compute the embedding vector $\mathbf{y}_i$ by minimizing the cost function in (23).

2.2.5 Independent component analysis

Independent component analysis (ICA) [32] is a statistical technique for separating a multidimensional random vector into linear additive subcomponents which are maximally statistically independent from each other. One of the most essential features of ICA model is nongaussianity, this feature is the key to reveal the independence of each components. Recently, ICA has gained much more attention in the filed of image processing, audio processing, biomedical signal processing and economics.

The goal of ICA is to maximize the statistical independence between the components of the basis vectors. Compared with other methods such as PCA, or PPCA, ICA searches for components which are both statistically independent and nongaussian. We can find the independent components by maximizing nongaussianity, it is an important principle in ICA estimation.

Normally, before implementing the ICA algorithm, the preprocessing steps of centering and whitening should be applied to make ICA estimation simpler and better conditioned.

We can use a statistical latent variables model to define ICA. For instance, we have observed the random vector $\mathbf{x} = (x_1, ..., x_m)^T$, and the independent component vector $\mathbf{s} = (s_1, ..., s_n)^T$. Therefore, the component x_i can be defined as

$$x_i = a_{i1}s_1 + a_{i2}s_2 + ...a_{ik}s_k + a_{in}s_n \tag{24}$$

here, a_{ik} are the mixing weights. If we use the matrix $\mathbf{A}$ to denote the weights of mixing components, then the generative ICA model can be defined as

$$\mathbf{x} = \mathbf{A}\mathbf{s} \, . \tag{25}$$

The independent component can be represented by

$$\mathbf{s} = \mathbf{W}\mathbf{x} \tag{26}$$

here, $\mathbf{W}$ is called the de-mixing matrix, and it is obtained by calculating the inverse of matrix $\mathbf{A}$.

Then, our job is to discover the mixing matrix $\mathbf{A}$ and the independent sources $\mathbf{s}$. This can be fulfilled by adaptively evaluating the value of $\mathbf{W}$ [32].

2.3 Experimental results

In our experiments, the process for generating new video textures contains three steps: the first step is to implement the dimensionality reduction techniques to extract the signatures from the frames of a input video. The second step is to apply an AR process to predict new frame signatures based on the signatures we obtained in the previous step. Here, we have a time series made of a sequence of frame signatures $\{\mathbf{x}_n\}$ where $n = 1, ..., N$, and N is the total number of frames. A zero-mean AR process of order p for a series of frame signatures in a d-dimensional space may be modelled by

$$\mathbf{x}_n = \sum_{k=1}^{p} A_k \mathbf{x}_{n-k} + \mathbf{w} \tag{27}$$

where the $d \times d$ matrices $A \equiv (A_1, ..., A_p)$ are the coefficient matrices of the AR process model. And $\mathbf{w}$ is a d-dimensional random vector drawn from Gaussian white noise distribution with zero-mean. In our case, the order p is set to 2, which means, the next frame signature will be generated based on the previous 2 signatures. The last step is to project these new frame signatures back to the image space and compose them together as a video texture. All of our experiments are done by using *Matlab* on a Windows platform.

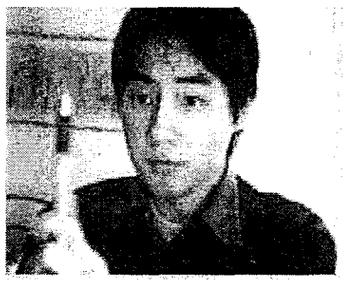


Figure 2.1: The original input video.

2.3.1 Extract Frame Signatures and Synthesize New Video Textures

In order to test the effect of choosing different dimension reduction techniques for creating video textures, we implemented several input videos.[2] For instance, one of the input videos we used is a video clip of a person moving a pen (as shown in Figure 2.1). It is 15 seconds long and contains 450 frames, and each frame has a size of 160 $\times$ 128 (20480) pixels . For each frame, the new image vector would be a vector with 20480 dimensions, and the total image matrix for 450 frames is a 450 $\times$ 20480 matrix, with images in the rows, and dimensions in the columns.

After acquiring the image matrix, we apply different dimensionality reduction techniques to capture the signatures of frames. In our experiments, the dimensionality of 30 to 40 is good enough for representing each individual frame. In the implementation of LLE and Isomap approaches, we use K-nearest method with the number of neighbors $K = 12$. In the case of applying ICA, we implemented the FastICA algorithm. A 'gaussian kernel' with the form $k(\mathbf{x}, \mathbf{x}') = \exp(- \parallel \mathbf{x} - \mathbf{x}' \parallel^2 /0.1)$ is implemented for the Kernel PCA. At the end of this step, each frame is represented by its signature which only has 30 dimensions. [3]

Subsequently, AR process is applied to synthesize new signatures. The number of new signatures is decided manually, for this input video, we set it to 60. At the end, 60 new frames are

[2] Some of our input test movies are obtained from the "Video Textures" web site:
http://www.cc.gatech.edu/cpl/projects/videotexture

[3] We have also applied mixtures of probabilistic principal component analysis [25] to generate the signatures of video frames, but the only good result we obtained is when the mixture size is equal to one which is same as the standard PPCA.

Table 2.1: Performances of Different dimension reduction techniques for generating video textures

Technique	Complexity	Run-time
PCA	$O(D^3)$	30.56
PPCA	$O(nD^2)$	38.17
Kernel PCA	$O(n^3)$	60.41
Isomap	$O(n^3)$	56.05
LLE	$O(Dn^2)$	45.49
ICA	$O(Dn^2)$	49.20

Table 2.2: The noise start to blur the result at which frame number

Dimensionality Reduction Techniques	**Frame Number**
PCA	16th
PPCA	50th
Kernel PCA	18th
Isomap	24th
LLE	20th
ICA	45th

synthesized and then define our video texture.

2.3.2 Comparison of the results

We have Implemented six different approaches (PCA, PPCA, Kernel PCA, Isomap, LLE and ICA) separately, all of them can produce good quality video textures. Each frame in the result is new and consists with the motions in the original video sequence. Figure 2.2 illustrates the first frame of the result generated by PCA, PPCA, kernel PCA, Isomap, LLE and ICA respectively. Table 2.1 demonstrates the computational complexity and the run-time for generating the new video texture. In Table 2.1, D is the dimensionality of the input data in the observed space, n represents the number of input data points and the unit for measuring run-time is in seconds. The Figure 2.3 shows the signatures extracted by applying different dimension reduction techniques.

Although the result seems very appealing, there still exist one problem, which is the occurrence of noise (i.e frames contain some 'ghost' in it as shown in Figure 2.4). For all the results, after

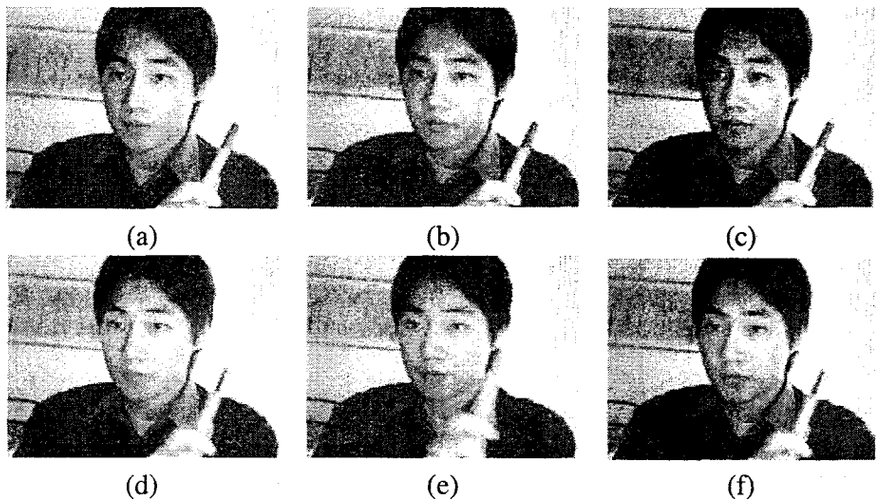


Figure 2.2: The first frame that is synthesized by using: (a) PCA , (b) PPCA, (c) kernel PCA, (D) Isomap, (e) LLE and (f) ICA.

some periods, the noise will start to become visible and make the video blur. The reason is that the AR process model predicts the new frame signature based on previous frames, the noise is cumulated as the AR process iteratively generates more new signatures. And this feature reflects the difference among PCA, PPCA, Kernel PCA, Isomap, LLE and ICA approaches for generating video textures. Table 2.2 demonstrates the noise start to appear at which frame in the synthesized video texture.

From Table 2.2, we may notice that for PCA, the noise starts to blur the frame at 16th frame, which is the earliest among all the six techniques. For PPCA and ICA, the noise appeared later than others (at 50th and 45th frames, respectively). That means, a more robust decomposition approach such as PPCA and ICA would generate more typical signature and therefore can improve the performance of the result. Among all the six approaches, PPCA provides the best result. This is reasonable, for PPCA, it is a latent variable model which conditioned on the observations data set, and it naturally reduces the dimensionality of the data from a probabilistic perspective. The order p of the AR process is another factor that may affect the performance of synthesizing video

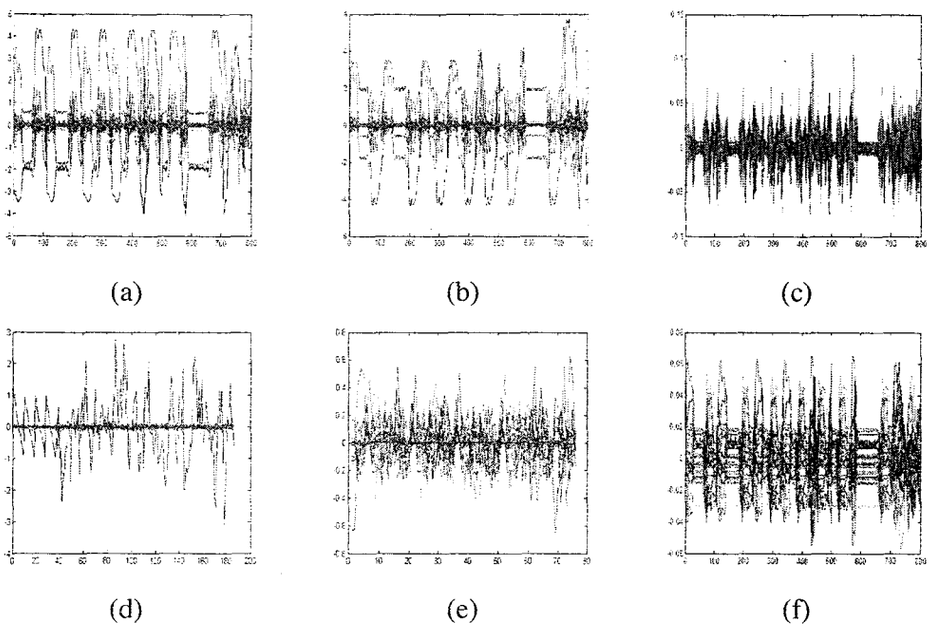


Figure 2.3: The signatures of frames extracted by (a) PCA , (b) PPCA, (c) Kernel PCA, (D) Isomap, (e) LLE and (f) ICA. The last 100 signatures are newly created by AR process. Here, the x-axis means the number of frames and the y-axis represents the value of coefficient of each input frame. Each color stands for one dimensionality of the frame signature (up to 30-dimensions).

Figure 2.4: The synthesized video frame which contains visible noise.

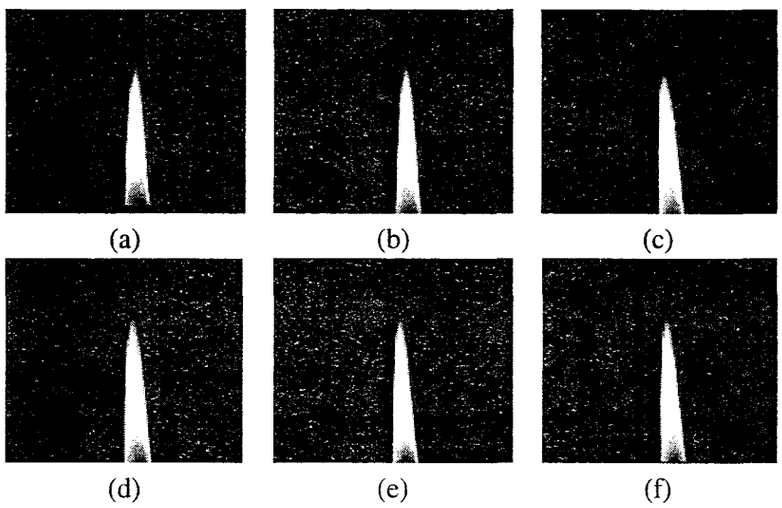


Figure 2.5: The first frame that is synthesized by using: (a) PCA , (b) PPCA, (c) Kernel PCA, (D) Isomap, (e) LLE and (f) ICA

textures. According to our experiments, as the value of p increases, the noise may blur the results earlier and faster. The reason is that, order p represents the amount of previous signatures that will be used for predicting a new signature, higher value of p will make the noise accumulated much faster. In our case, the best results are obtained when p is set to 2.

2.3.3 More experimental results

We have also experimented other movies during our study. Here, the goal is to test the video texture generation approach on some structurally complex scenarios such as fountain. And we also investigated to use it on some cartoon movies, since compared to the real shooted movie, cartoon movies contain more sparse space and less color intensity. Figure 2.5 shows the synthesized result from a movie which contains a movement of a burning candle for each dimensionality reduction technique. In Figure 2.6, we illustrate the generated video textures for a cartoon animation. Figure 2.7 demonstrates the results of the video textures for a movie of fountain. Figure 2.8 and

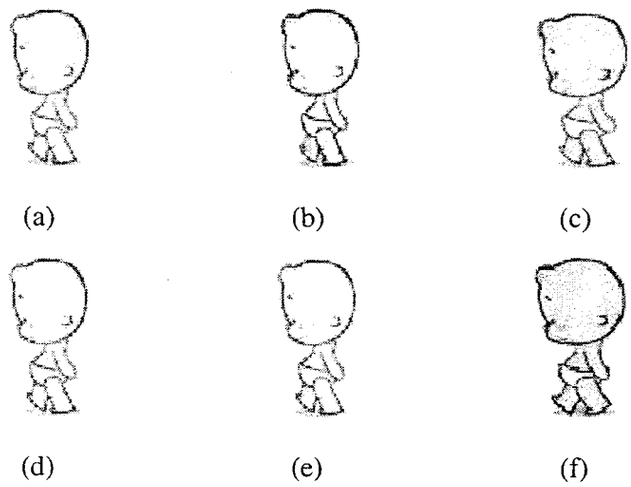


Figure 2.6: The first frame that is synthesized by using: (a) PCA , (b) PPCA, (c) Kernel PCA, (D) Isomap, (e) LLE and (f) ICA for an animation of cartoon.

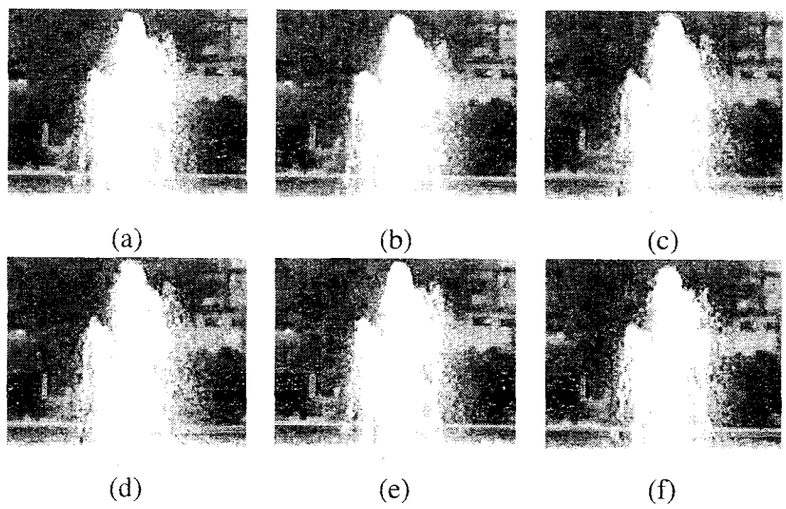


Figure 2.7: The first frame that is generated by using: (a) PCA , (b) PPCA, (c) Kernel PCA, (D)Isomap, (e) LLE and (f) ICA for a movie of fountain.

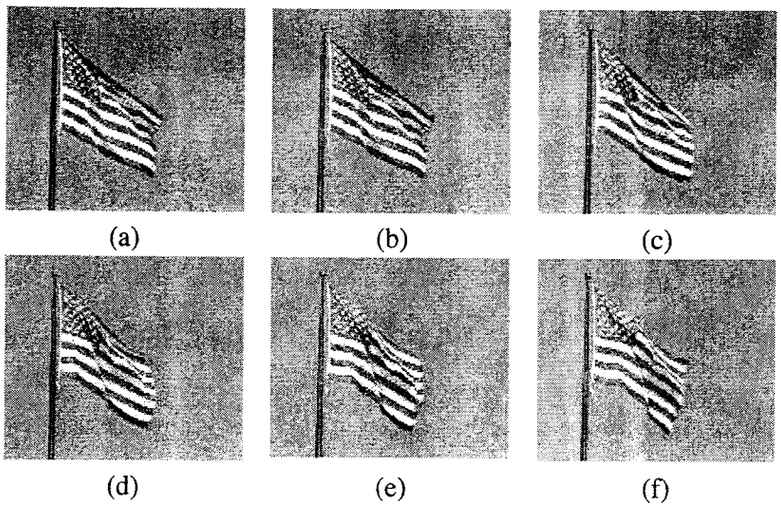


Figure 2.8: The first frame that is generated by using: (a) PCA , (b) PPCA, (c) Kernel PCA, (D)Isomap, (e) LLE and (f) ICA for a movie of flag.

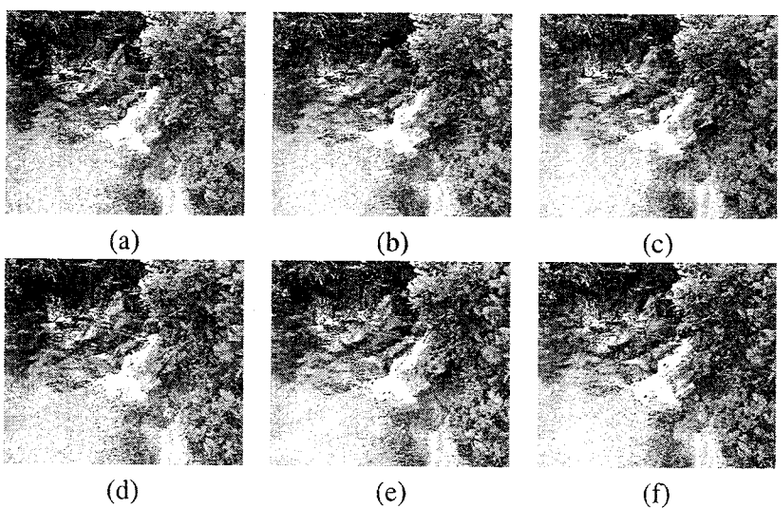


Figure 2.9: The first frame that is generated by using: (a) PCA , (b) PPCA, (c) Kernel PCA, (D)Isomap, (e) LLE and (f) ICA for a movie of waterfall.

Table 2.3: The noise start to blur the result at which frame number for different input movies and techniques)

Movie	PCA	PPCA	K-PCA	Isomap	LLE	ICA
Flame (Figure 2.5)	17th	47th	21th	20th	18th	38th
Cartoon (Figure 2.6)	20th	34th	24th	38th	35th	30th
Fountain (Figure 2.7)	13th	39th	15th	22th	24th	35th
Flag (Figure 2.8)	17th	40th	19th	25th	22th	34th
Waterfall (Figure 2.9)	14th	36th	15th	24th	23th	33th

Figure 2.9 show the video texture results of a movie of waving flag and a waterfall, respectively. Tables 2.3 demonstrates at which frame number the noise start to become visible for the flame, cartoon, fountain, flag and waterfall movies, respectively. Here, from Table 2.3 we may notice that, for cartoon movie, IsoMap and LLE provide better solution. This is because a cartoon movie are more spare compared to the real movies, and dimensionality reduction techniques based on preserving the neighborhood relationship such as Isomap and LLE are more suitable than others [4].

[4]ST-IsoMap is also tested for cartoon movies, it can generate a better results than all other results. A work based on using ST-IsoMap to generate cartoon textures can be found in [33].

Chapter 3

Generate Video Textures by PPCA and Gaussian Process Dynamical Model

In the previous chapter, we presented the approach of synthesizing video textures by dimensionality reduction techniques and AR process. This approach is efficient and can provide good quality results. However, the synthesized video textures still contain noise and this may blur the video after some periods. Therefore, discovering a model which is more robust to noise is our next task. In this chapter, we propose a new method of generating video textures by implementing probabilistic principal components analysis (PPCA) and Gaussian Process Dynamical models (GPDMs). Compared to the original video texture technique, video textures synthesized by PPCA and GPDM has the following advantages: it may generate new frames that have never existed in the input video clip, the problem of "dead-end" is totally avoided, and it can also provide video textures that are more robust to noise.

3.1 Introduction and Related Works

Recently, a number of extensions and applications of video texture have emerged. In [34], the authors propose a novel method of generating video texture based on wavelet transform. Instead of creating video textures by simply changing the order of frames in the input video clip, they construct the new frames based on wavelet coefficients which are computed from the decomposition of the pixel values of neighboring frames. In [23], video texture is synthesized first by applying the

principal components analysis (PCA) to obtain the signatures of each frame, then autoregressive process (AR) is used to predict new frames.

Our work is inspired by [23], where the authors have shown that video texture may be created by implementing regression methods such as Autoregressive Process (AR) which allow the prediction of new video frames. Accordingly, new video textures are obtained by appending synthesized frames. Gaussian process [35] [36] [37] is another approach which can be exploited to solve regression problems. Via Gaussian process, we can define probability distributions over functions directly. We can combine a Gaussian process prior with a likelihood to acquire a posterior over functions.

In our work, we adopt an extension of Gaussian process namely Gaussian process dynamical model (GPDM) [38] [39]. It is a latent variable model that can be applied for nonlinear time series analysis. Its idea is derived from the Gaussian process latent variable model (GPLVM) [40]. However, GPLVM is not a dynamical model, it considers that data points in the data set independent, and it ignores the continuity of data points. The GPDM augments the GPLVM with a latent dynamical model. It includes a low-dimensional space account for dynamics in the time series data, as well as a mapping from the latent space to observation space. It can make predictions about future data and also consider the temporal dependence between the data sets. Since video sequence is a time series data, in principle, GPDM is a suitable method to synthesize new video textures. Therefore, we implemented GPDM on serval video clips to synthesize new video textures and compared the results with video textures generated by AR process. The authors in [41] have applied PCA as a dimensionality reduction technique to obtain the frames signatures. However, we have shown in a previous works [42] that probabilistic principal components analysis (PPCA) [24] is more robust to noise and provide better results. Thus, our video texture generation framework will be based on both PPCA and GPDMs.

The remainder of this chapter is organized as follows. First we introduce Gaussian processes in Section 3.2. Then GPDM for video texture is discussed in Section 3.3. Section 3.4 is devoted to the experimental results.

3.2 Gaussian Processes

A Gaussian process is defined as a probability distribution over some functions $y(\mathbf{x})$, such that the set of values of $y(\mathbf{x})$ evaluated at an arbitrary set of points $\mathbf{x}_1, ..., \mathbf{x}_N$ jointly have a Gaussian distribution. Although it may seems impossible to work with uncountably infinite space of functions at the first view, we only need to consider the values of the function at input data points $\mathbf{x}_n$. Gaussian process can also be applied on classification problem, but we will not discuss it here.

3.2.1 Linear regression in function space

In order to illustrate Gaussian process, we can first illustrate a particular example of Gaussian process. It is provided by analyzing a linear regression model that its predictive distribution is defined by working over functions. Let us define a model as

$$y(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) \tag{1}$$

where $\mathbf{x}$ is an input vector, $\mathbf{w}$ is an M-dimensional weight vector and $\phi(\mathbf{x})$ are the fixed basis functions for input $\mathbf{x}$. The prior distribution over $\mathbf{w}$ is an isotropic Gaussian

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|0, \alpha^{-1}\mathbf{I}) \tag{2}$$

where the hyperparameter α is the precision of the distribution. From (1), we know that any value of $\mathbf{w}$ defines a specific function of $\mathbf{x}$, therefore, the prior distribution of $\mathbf{w}$ in (2) implies a probability distribution over function $y(\mathbf{x})$. We are interested in discovering the joint distribution of function values $y(\mathbf{x}_1), ..., y(\mathbf{x}_N)$ given the data points $\mathbf{x}_1, ..., \mathbf{x}_N$. Then, we can rewrite (1) as

$$\mathbf{y} = \mathbf{\Phi w} \tag{3}$$

where $\mathbf{y}$ is a vector containing the values of $y_1, ..., y_N$ and $\mathbf{\Phi}$ denotes the $N \times M$ design matrix with elements $\mathbf{\Phi}_{nk} = \phi_k(\mathbf{x}_n)$. Since $\mathbf{y}$ is a linear combination of $\mathbf{w}$ and $\mathbf{w}$ is a Gaussian distribution, thus

$\mathbf{y}$ is also Gaussian. Therefore, we can obtain the probability distribution of $\mathbf{y}$ by just computing its mean $E[\mathbf{y}]$ and covariance $cov[\mathbf{y}]$. They are calculated as

$$E[\mathbf{y}] = \mathbf{\Phi} E[\mathbf{w}] = 0 \tag{4}$$

$$cov[\mathbf{y}] = E[\mathbf{y}\mathbf{y}^T] = \mathbf{\Phi} E[\mathbf{w}\mathbf{w}^T]\mathbf{\Phi}^T = \frac{1}{\alpha}\mathbf{\Phi}\mathbf{\Phi}^T = \mathbf{K} \tag{5}$$

where $\mathbf{K}$ is the Gram matrix and its elements can be represented through a kernel function $k(\mathbf{x}, \mathbf{x}')$ as

$$K_{nm} = k(\mathbf{x}_n, \mathbf{x}_m) = \frac{1}{\alpha}\phi(\mathbf{x}_n)^T\phi(\mathbf{x}_m) \ . \tag{6}$$

The model above is a specific example of Gaussian process. In Gaussian processes, the joint probability distribution of function values $y_1, ...y_N$ over N variables is fully defined by the mean and covariance. In general, we will not have any prior knowledge about the mean, so for simplicity we choose it to be zero. Then, the Gaussian process is fully defined by its covariance which is the kernel function.

3.2.2 Gaussian process regression

In the last section, we have seen a specific example of Gaussian process given by defining the predictive probability distribution of a linear regression model over function space. Here, we will illustrate how Gaussian process can be applied for general regression problems. We consider a model where the observed target values t_n are corrupted with some random noise

$$t_n = y_n + \epsilon_n \tag{7}$$

where $y_n = y(\mathbf{x}_n)$ for input data $\mathbf{x}$. ϵ_n is the random noise which has Gaussian noise with zero mean and β^{-1} variance

$$\epsilon \sim \mathcal{N}(0, \beta^{-1}) \ . \tag{8}$$

β is the precision (inverse variance) of the noise distribution. Then the probability distribution of the observed target values is also Gaussian

$$p(t_n|y_n) = \mathcal{N}(t_n|y_n, \beta^{-1}) \ . \tag{9}$$

Since the noise is independent for each data point, given the values of $\mathbf{y} = (y_1, ...y_N)^T$, the joint distribution of target values $t = (t_1, ..., t_N)$ is an isotropic Gaussian

$$p(\mathbf{t}|\mathbf{y}) = \mathcal{N}(\mathbf{t}|\mathbf{y}, \beta^{-1}\mathbf{I}_N) \tag{10}$$

where $\mathbf{I}_N$ is the $N \times N$ unit matrix. From (5) in the last section, we know that in Gaussian process, the marginal distribution of $p(\mathbf{y})$ is a Gaussian with zero mean and the covariance is defined by a Gram matrix $\mathbf{K}$ as

$$p(\mathbf{y}) = \mathcal{N}(\mathbf{y}|0, \mathbf{K}) \tag{11}$$

here, $\mathbf{K}$ is determined by some kernel functions $k(\mathbf{x}_n, \mathbf{x}_m)$. This represents how strong the correlation is between two target values. For instance, if two points $\mathbf{x}_n$ and $\mathbf{x}_m$ are similar, that means the values of y_n and y_m are strongly correlated, and vise versa.

In order to make predictions with this model, first we need to calculate the marginal distribution of $p(\mathbf{t})$. That is, given the input values of $\mathbf{x}_1, ...\mathbf{x}_N$, we need to integrate over $\mathbf{y}$. The result of the marginal distribution of $\mathbf{t}$ is

$$p(\mathbf{t}) = \int p(\mathbf{t}|\mathbf{y})p(\mathbf{y})d\mathbf{y} = \mathcal{N}(\mathbf{t}|0, \mathbf{C}) \tag{12}$$

where the elements of covariance matrix $\mathbf{C}$ are given by

$$C(\mathbf{x}_n, \mathbf{x}_m) = k(\mathbf{x}_n, \mathbf{x}_m) + \beta^{-1}\delta_{nm} \tag{13}$$

where δ_{nm} is the Kronecker delta which is qual to one iff $m = n$ and zero, otherwise.

After obtaining the marginal distribution of $\mathbf{t}$, the next job is to evaluate the conditional distribution $p(t_{N+1}|\mathbf{t})$ where t_{N+1} is the next target value that we wish to predict. In order to find $p(t_{N+1}|\mathbf{t})$, we first need to find the joint distribution of $p(\mathbf{t}_{N+1})$ for $t_1, ..., t_{N+1}$, this can be done from (12)

$$p(\mathbf{t}_{N+1}) = \mathcal{N}(\mathbf{t}_{N+1}|0, \mathbf{C}_{N+1}) \tag{14}$$

where $\mathbf{C}_{N+1}$ is an $(N+1) \times (N+1)$ covariance matrix.

Since the joint distribution $p(\mathbf{t}_{N+1})$ is also a Gaussian distribution, our next goal is to compute the predictive distribution $p(t_{N+1}|\mathbf{t})$. The covariance matrix $\mathbf{C}_{N+1}$ needs to be partitioned as

$$\mathbf{C}_{N+1} = \begin{pmatrix} \mathbf{C}_N & \mathbf{k} \\ \mathbf{k}^T & c \end{pmatrix} \tag{15}$$

where $\mathbf{C}_N$ is the $N \times N$ covariance matrix of the training data, vector $\mathbf{k}$ represents the $N \times 1$ covariance matrix of training data and the predictive target t_{N+1}, and the scalar c denotes the variance of t_{N+1}. We can obtain the mean and covariance of the conditional distribution $p(t_{N+1}|\mathbf{t})$ in the form of

$$m(\mathbf{x}_{N+1}) = \mathbf{k}^T \mathbf{C}_N^{-1} \mathbf{t} \ . \tag{16}$$

$$\sigma^2(\mathbf{x}_{N+1}) = c - \mathbf{k}^T \mathbf{C}_N^{-1} \mathbf{k} \ . \tag{17}$$

These results represent the core idea of Gaussian process regression. More details and discussions about Gaussian processes can be found in [43].

3.3 Gaussian Processes Dynamical Models

The Gaussian process dynamical model (GPDM) [39] is a latent variable model, it has two nonlinear mappings. One mapping is from the latent space to the observation space, the other is the

dynamical mapping in the latent space. In the GPDM, these two mappings are learned from Gaussian process regression. Because of the properties for the Gaussian process regression, the GPDM does not need to select many parameters in order to learn nonlinear dynamical models.

Suppose $\{\mathbf{y}_1, ..., \mathbf{y}_N\}$ denotes the D-dimensional observation data set and $\mathbf{y}_t$ represents a particular observation output at the specific time t, $\mathbf{y}_t \in \mathbb{R}^D$. $\mathbf{x}_1, ..., \mathbf{x}_N$ is a data set in the latent space, $\mathbf{x}_t$ represents the d-dimensional latent coordinate of the observation data at time index t, $\mathbf{x}_t \in \mathbb{R}^d$. $\mathbf{A}$ and $\mathbf{B}$ are parameters of the dynamical and observation mapping functions, respectively. In order to generate the GPDM, the parameters in the two mappings need to be marginalized out, and the latent coordinates of the training data need to be optimized. First we expect to evaluate the joint probability distribution of $\mathbf{Y} = \mathbf{y}_1, ..., \mathbf{y}_N$. The vector $\mathbf{x}$ repents a data point in the latent space where the dimensionality is much lower than the observation space. The first-order Markov dynamics is used to represent the dynamical system in the latent space

$$\mathbf{x}_t = f(\mathbf{x}_{t-1}; \mathbf{A}) + \mathbf{n}_{x,t} \,. \tag{18}$$

In the above equation, $\mathbf{x}_t \in \mathbb{R}^d$, it shows the d-dimensional latent coordinates at time t, the dynamical mapping function f is parameterized by $\mathbf{A}$, and $\mathbf{n}_{x,t}$ is a zero-mean, isotropic, white Gaussian noise process. The mapping from the latent space to the observation space is

$$\mathbf{y}_t = g(\mathbf{x}_t; \mathbf{B}) + \mathbf{n}_{y,t} \tag{19}$$

here, the function g is parameterized by $\mathbf{B}$, and same as $\mathbf{n}_{x,t}$, $\mathbf{n}_{y,t}$ is a zero-mean, isotropic, white Gaussian noise process.

The task now is to infer the function from the given data, in a parametric approach for regression, functions f and g can be expressed in terms of two nonlinear functions $f(x; \mathbf{A})$ and $g(x; \mathbf{B})$ parameterized by parameters $\mathbf{A}$ and $\mathbf{B}$, respectively. Two basis functions ϕ_i and φ_j are used for f and g

$$f(\mathbf{x}; \mathbf{A}) = \sum_i \mathbf{a}_i \phi_i(\mathbf{x}) \tag{20}$$

$$g(\mathbf{x};\mathbf{B}) = \sum_j \mathbf{b}_j \varphi_i(\mathbf{x}) \tag{21}$$

where weights $\mathbf{A} \equiv [a_1, a_2, ...]^T$ and $\mathbf{B} \equiv [b_1, b_2, ...]^T$. f and g are nonlinear functions of $\mathbf{x}$, but the dependencies of f and g on the parameters $\mathbf{A}$ and $\mathbf{B}$ are linear. To infer this model with the training data, the parameters $\mathbf{A}$ and $\mathbf{B}$ must be estimated and the number of basis functions must be appropriately selected. It is also required to ensure that there is enough data to constrain the shape of each basis function. In the Bayesian view, because the model parameters $\mathbf{A}$ and $\mathbf{B}$ are very uncertain, for the GPDM, the parameters should be marginalized out if possible.

3.3.1 Latent Space Mapping

For the mapping from latent space to the observation space, function g in (21) is a linear combination of columns of $\mathbf{B}$, where $\mathbf{B}$ has an isotropic Gaussian prior. After marginalizing over g, the joint distribution of $\mathbf{Y}$ can be represented as

$$p(\mathbf{Y}|\mathbf{X}, \bar{\beta}, \mathbf{W}) = \frac{|\mathbf{W}|^N}{\sqrt{(2\pi)^{ND}|\mathbf{K}_Y|^D}} \exp(-\frac{1}{2} tr(\mathbf{K}_Y^{-1}\mathbf{Y}\mathbf{W}^2\mathbf{Y}^T)) \,. \tag{22}$$

This denotes the probability of the observations $\mathbf{Y}$ given the latent coordinates $\mathbf{X}$ and hyperparameters $\bar{\beta}$ and $\mathbf{W}$. $\mathbf{K}_Y$ is the kernel matrix of the mapping g and $\bar{\beta}$ are the hyperparameters of the kernel. $\mathbf{W}$ represents the scale parameters which account for the overall scale in each data dimension. This is because for many data sets, different data dimensions have different variances. The scaling parameters $\mathbf{W} \equiv diag(w_1, ..., w_D)$ are used to model the variance in each observation dimension. The elements of $\mathbf{K}_Y$ are defined by a kernel function $(K_Y)_{ij} \equiv \mathbf{k}_Y(\mathbf{x}_i, \mathbf{x}_j)$. We choose the radial basis function (RBF) as the kernel function for the latent mapping g

$$k_Y(\mathbf{x}, \mathbf{x}') = \beta_1 \exp(-\frac{\beta_2}{2}\|\mathbf{x} - \mathbf{x}'\|^2) + \beta_3^{-1}\delta_{\mathbf{x}.\mathbf{x}'} \tag{23}$$

here, the hyperparameter β_1 represents the output scale of the kernel function, and β_2 represents the inverse width of the RBF. β_3 gives the variance of the isotropic noise term $\mathbf{n}_{y,t}$.

3.3.2 Dynamic Mapping

The dynamic mapping for latent coordinate is similar to the latent space mapping. The weights $\mathbf{A}$ are marginalized for the joint probability density over the latent coordinates

$$p(\mathbf{X}|\bar{\alpha}) = \int p(\mathbf{X}, \mathbf{A}|\bar{\alpha})d\mathbf{A} = \int p(\mathbf{X}|\mathbf{A}, \bar{\alpha})p(\mathbf{A}|\bar{\alpha})d\mathbf{A} \ . \tag{24}$$

In the above equation, $\bar{\alpha}$ is a vector of hyperparameters for the kernel. If the first-order Markov dynamical property in (18) is considered, the result is

$$p(\mathbf{X}|\bar{\alpha}) = p(\mathbf{x}_1) \int \prod_{t=2}^{N} p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{A}, \bar{\alpha})p(\mathbf{A}|\bar{\alpha})d\mathbf{A} \ . \tag{25}$$

Just like in the latent space mapping, we take an isotropic Gaussian prior on the columns of the weights $\mathbf{A}$, then (25) can be simplified to the form

$$p(\mathbf{X}|\bar{\alpha}) = \frac{p(\mathbf{x}_1)}{\sqrt{(2\pi)^{(N-1)d}|\mathbf{K}_X|^d}} \exp(-\frac{1}{2}tr(\mathbf{K}_X^{-1}\mathbf{X}_{2:N}\mathbf{X}_{2:N}^T)) \ . \tag{26}$$

In the above equation, it is assumed that $\mathbf{x}_1$ also has a Gaussian prior, and $\mathbf{X}_{2:N} = [\mathbf{x}_2, ...\mathbf{x}_N]^T$ denotes the input data that except the first element. $\mathbf{K}_X$ is the kernel matrix build from $[\mathbf{x}_1, ...\mathbf{x}_{N-1}]$. In this dynamic mapping, the form "RBF + linear" is defined for the kernel function

$$k_X(\mathbf{x}, \mathbf{x}') = \alpha_1 \exp(-\frac{\alpha_2}{2}\|\mathbf{x} - \mathbf{x}'\|^2) + \alpha_3 x^T x' + \alpha_4^{-1}\delta_{\mathbf{x},\mathbf{x}'} \tag{27}$$

here the RBF term is used to model nonlinear dynamics, and the linear term is used to make the system to regress to linear dynamics when predictions are made from the training data. α_1 and α_3 is the output scale parameter of the RBF term and linear term, respectively, and α_2 represents inverse width of the RBF term. α_4^{-1} represents the variance of the noise term $n_{x,t}$ in (18).

3.3.3 Priors On Hyperparameters

In order to reduce the chance of overfitting in the learning process, a principled way is to attach prior distributions on the parameters, so that certain values of the parameters are preferred to others

that may cause overfitting. According to the GPLVM, uninformative priors are placed on the kernel hyperparameters $\bar{\alpha}$ and $\bar{\beta}$ as follows

$$p(\bar{\alpha}) \propto \prod_i \alpha_i^{-1} . \tag{28}$$

$$p(\bar{\beta}) \propto \prod_i \beta_i^{-1} . \tag{29}$$

These priors assign high probabilities to small values of $\bar{\alpha}$ and $\bar{\beta}$. Small values of α_2 and β_2 denote a preference for a large width of the RBFs. Small values of α_1 and β_1 represent small output scales and small values of α_4 and β_3 represent large noise variances. For the hyperparameter $\mathbf{W}$, a broad half-normal prior is placed on it

$$p(\mathbf{W}) = \prod_{m=1}^{D} \frac{2}{k\sqrt{2\pi}} \exp(-\frac{w_m^2}{2k^2}) \tag{30}$$

where w_m are the variances that contain the elements of $\mathbf{W}$. In our implementation, a value 10^3 is assigned to k. This prior shows that on every dimension of the data, there is a nonzero variance. Then a generative model for time-series observations can be obtained through the latent space mapping, the dynamic mapping and the priors

$$p(\mathbf{X}, \mathbf{Y}, \bar{\alpha}, \bar{\beta}, \mathbf{W}) = p(\mathbf{Y}|\mathbf{X}, \bar{\beta}, \mathbf{W}) p(\mathbf{X}|\bar{\alpha}) p(\mathbf{W}) p(\bar{\alpha}) p(\bar{\beta}) . \tag{31}$$

This represent the general form of the GPDM. The details of how does GPDM work can can found in [38].

3.3.4 Learning the parameters

In order to learn the GPDM from an observation data set $\mathbf{Y}$, the unknowns in the model ($\mathbf{X}, \bar{\alpha}, \bar{\beta}$, $\mathbf{W}$) need to be estimated using numerical optimization method. An intuitive method is to apply maximum a posteriori (MAP) algorithm for estimating all the unknowns ($\mathbf{X}, \bar{\alpha}, \bar{\beta}, \mathbf{W}$)) up to an

additive constant. For simplicity, it is same as minimizing the negative log-posterior of the unknowns

$$L = -\ln p(\mathbf{X}, \bar{\alpha}, \bar{\beta}, \mathbf{W}|\mathbf{Y}) \ . \tag{32}$$

Then, it gives

$$L = L_Y + L_X + \sum_j \ln \beta_j + \sum_j \ln \alpha_j + \frac{1}{2k^2} tr(\mathbf{W}^2) \tag{33}$$

where

$$L_Y = \frac{D}{2} \ln |\mathbf{K}_Y| + \frac{1}{2} tr(\mathbf{K}_Y^{-1} \mathbf{Y} \mathbf{W}^2 \mathbf{Y}^T) - N \ln |\mathbf{W}| \tag{34}$$

and

$$L_X = \frac{d}{2} \ln |\mathbf{K}_X| + \frac{1}{2} tr(\mathbf{K}_X^{-1} \mathbf{X}_{2:N} \mathbf{X}_{2:N}^T) + \frac{1}{2} \mathbf{x}_1^T \mathbf{x}_1 \ . \tag{35}$$

For learning the unknown parameters $\{\mathbf{X}, \bar{\alpha}, \bar{\beta}, \mathbf{W}\}$, first the input data is mean-subtracted, and the latent coordinates are initialized with dimensionality reduction techniques (e.g. PCA or PPCA). Then L is minimized with respect to parameter $\mathbf{W}$ in closed form: $w_k \Leftarrow \sqrt{N(\mathbf{y}_{:,k}^T \mathbf{K}_Y^{-1} \mathbf{y}_{:,k} + \frac{1}{k^2})^{-1}}$ and with respect to the parameters $\{\mathbf{X}, \bar{\alpha}, \bar{\beta}\}$ by using the scaled conjugate gradient (SCG) method [36]. This algorithm will be iterated until an additive constant is found. Then, we will obtain these unknown parameters.

3.4 Experimental Results

In our work, the goal is to apply GPDM to synthesize video textures. The performance of our approach is evaluated by comparing our results with the video textures generated by AR approach in [41]. In the AR approach for synthesizing video textures, frame signatures are first calculated by adopting the dimension reduction technique: principal components analysis (PCA), followed by the synthesis of new video textures using AR process. In our case, in order to test our approach under different scenarios, several input video clips are selected. First, the input video clip is decomposed into a sequence of frames. Each individual frame is an input vector $\mathbf{x}$, with dimensionality D. The value of D is the number of pixels contained in each frame. Second, these input

vectors are mean-subtracted and the latent coordinates are initialized with PPCA. Last, GPDM is applied to synthesize new video frames which are then composed together to generate a new video texture.

3.4.1 Generation of new frames

As described above, new video frames are predicted using GPDM. In other words, it is to predict the next video frame $\mathbf{x}_{N+1}$ conditioned on the previous frame $\mathbf{x}_N$. The marginal distribution of the new frame $p(\mathbf{x}_{N+1})$ derived from the conditional distribution $p(\mathbf{x}_{N+1}|\mathbf{x}_N)$ is also a Gaussian distribution

$$\mathbf{x}_{N+1} \sim \mathcal{N}(\mu_X(\mathbf{x}_N); \sigma_X^2(\mathbf{x}_N)) \ . \tag{36}$$

We can solve this prediction problem by applying the similar ideas as in Gaussian process regression. According to results in (16) and (17), the mean and covariance can be calculated as

$$\mu_X(\mathbf{x}) = \mathbf{X}_{2:N}^T \mathbf{K}_X^{-1} \mathbf{k}_X(\mathbf{x}) \ . \tag{37}$$

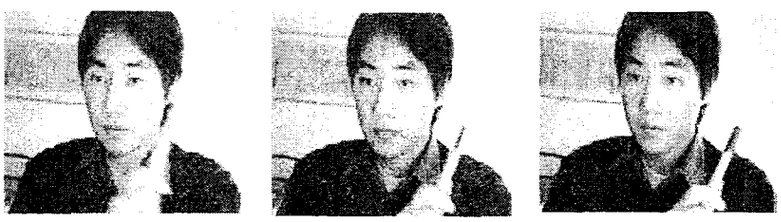
$$\sigma_X^2(\mathbf{x}) = k_X(\mathbf{x}, \mathbf{x}) - \mathbf{k}_X(\mathbf{x})^T \mathbf{K}_X^{-1} \mathbf{k}_X(\mathbf{x}) \ . \tag{38}$$

In the above equations, $\mathbf{k}_X(\mathbf{x})$ represents a vector that contains the covariance $\mathbf{k}_X(\mathbf{x}, \mathbf{x}_i)$ in the i-th entry and $\mathbf{x}_i$ denotes the i-th training vector. Then, the next frame in the latent space is: $\mathbf{x}_{N+1} = \mu_X(\mathbf{x}_N)$. Therefore, the new video frames can be generated by $\mathbf{y}_{N+1} = \mu_Y(\mathbf{x}_{N+1})$.

New video textures are successfully generated from input video clips by applying PPCA and GPDM with 50 frames in each video texture. They can be played without any visual discontinuity but with similar motions as the original one. Moreover, all resulted frames have never appeared before in the input videos. Figure 3.1 $\sim$ Figure 3.6 demonstrate the first three frames generated by PPCA and GPDM.

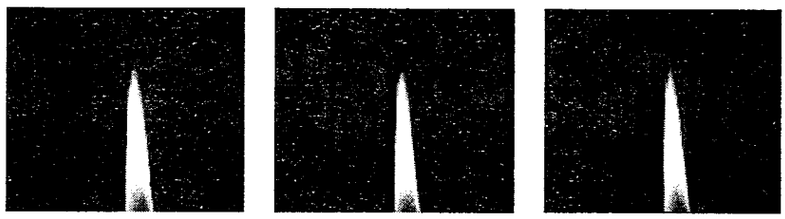
3.4.2 Comparison of the results

In this section, we compare the performance of synthesizing video textures by GPDM and AR process. Via the AR process, although the result seems very good, there is still one problem which is the occurrence of noise. For all results, after a certain time, the noise will start to become visible and make the video blur. However, through GPDM, it is more robust to noise compared to AR process since it contains a latent space account for the dynamics in the input data. As shown in Figure 3.7, the 20th, 25th and 30th frames generated by PCA and AR process contain much more noise than the ones produced by PPCA and GPDM at each corresponding frame number. Based on our experimental results, we may conclude that video textures generated by PPCA and GPDM can provide better results with more robustness to noise than AR approach. The synthesized new video textures contain similar motions as the input video clips and all frames in the new video textures are completely new.



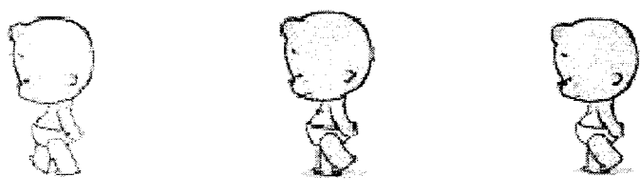
(a) (b) (c)

Figure 3.1: The first three frames are synthesized by PPCA and GPDM.



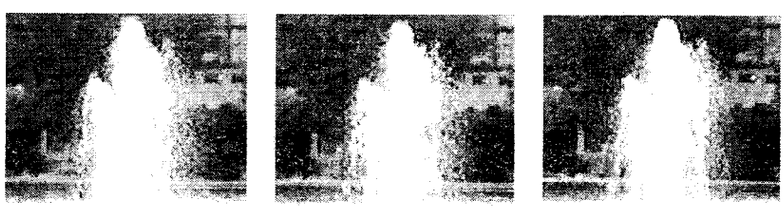
(a) (b) (c)

Figure 3.2: The first three frames are synthesized by PPCA and GPDM.



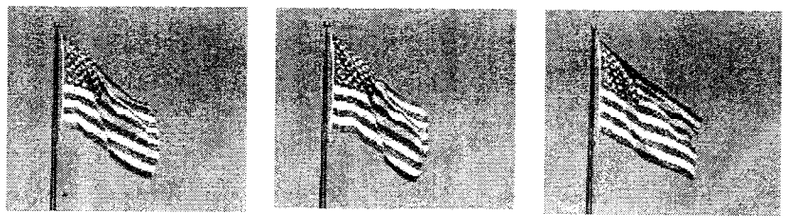
(a) (b) (c)

Figure 3.3: The first three frames are synthesized by PPCA and GPDM.



(a) (b) (c)

Figure 3.4: The first three frames are synthesized by PPCA and GPDM.



(a) (b) (c)

Figure 3.5: The first three frames are synthesized by PPCA and GPDM.

(a) (b) (c)

Figure 3.6: The first three frames synthesized by PPCA and GPDM.

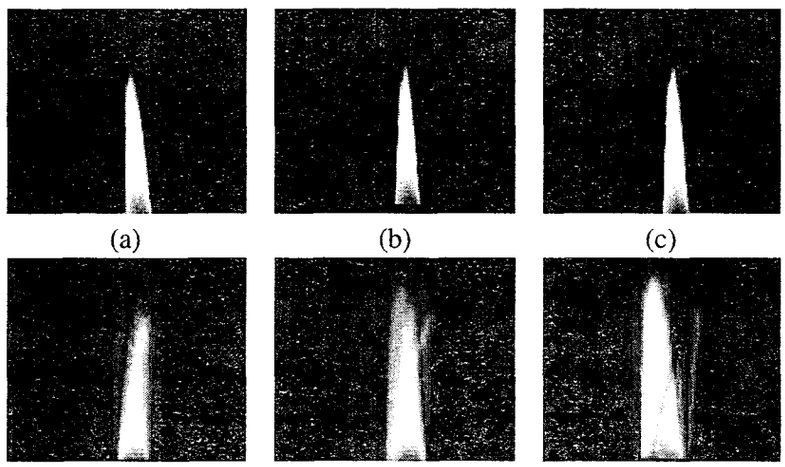


Figure 3.7: (a), (b) and (c) illustrate the 20th, 25th and 30th frames synthesized by PPCA and GPDM; (d), (e) and (f) demonstrate the 20th, 25th and 30th frames generated by PCA and AR process.

Chapter 4

Generate Real-time Video Textures by Incremental Isomap and Incremental ST-Isomap

Recently, some new approaches for generating video textures have been proposed. All these approaches operate in a batch mode, meaning that all the input data points (input video frames in our case) have to be available during the generating process. Batch approach works well in most cases. However, in some applications, such as surveillance, since data are observed sequentially, batch methods are computationally demanding and take a longer time to generate a new video texture. In this chapter, we propose two different approaches for synthesizing real-time video textures. The first one applies incremental Isomap and Autoregressive (AR) process to generate real-time video textures, which is applicable for most videos. Nonetheless, it does not extend well for videos that are more sparse, such as cartoons. Inspired by spatio-temporal Isomap (ST-Isomap), we propose an extension of incremental Isomap which contains spatio-temporal coherence in the data set and can also be applied in incremental mode. We name it as Incremental spatio-temporal Isomap (IST-Isomap). Instead of applying incremental Isomap, our second real-time video texture generation approach exploits IST-Isomap and AR process. It provides more efficient and better quality results for sparse videos than the first approach. Compared with other video texture generation approaches, both of our approaches are able to synthesize new video textures in a real-time fashion which in turn offer advantages (e.g. faster and more efficient) in applications where data are sequentially obtained.

4.1 Introduction and Related Works

Video texture introduced by Schödl *et al.* [5] is a new type of medium that can generate a continuous, infinitely changing stream of images from a recorded video. Furthermore, more applications related to video textures may be found in [9] [8] [21] [13]. A similar idea as video texture, "dynamic texture", proposed by Doretto *et al.* [44], represents sequences of images that exhibit the same form of temporal stationarity, such as waves, smoke, *etc.* Besides, dynamic textures are able to use a small number of resource data to generate a larger or even infinite number of new data. Doretto and coworkers first built a statistical model which accounts for the intrinsic dynamics in the input data; then they optimized this model by maximizing likelihood or minimizing the prediction error variance. More related works and details can be found in [45] [46].

To the best of our knowledge, all of the video texture generation methods so far work in a batch mode, which means, all the frame data have to be collected before the generation process starts. The drawback of the batch video texture methods is that, in some situations where the training images become available sequentially such as surveillance, the computations become prohibitive. Another problem is that, in order to update the lower dimensional subspace for frame 'signatures' with a new observed image, we have to recompute the low-dimensional representation for the whole data set.

In this chapter, we propose two different approaches that allow video textures to be generated in a real-time fashion. Our work is inspired by Fitzgibbon's work [41], where dimension reduction technique and AR process were implemented to synthesize new video textures. In our case, we extend Fitzgibbon's method by applying incremental Isomap (recently proposed in [47]) and AR process [22] to establish a real-time version of video texture generation process. This approach produces good results for most videos except for comparable sparse video data such as cartoons. The reason is that the cartoon data are more inherently spare and also contain exaggerated deformations between frames. Juan and coworkers [33] applied spatio-temporal Isomap (ST-Isomap) [48] to generate better results for cartoon textures than original Isomap since it maintains the temporal

coherence in the data set while finding the low-dimensional embedding. Thus, ST-Isomap is appropriated for dealing with cartoon data. However, it still works in a batch mode. Based on the idea of incremental Isomap, we propose an extension of ST-Isomap which contains spatio-temporal coherence in the data set and works in real-time. We name it as incremental spatio-temporal Isomap (IST-Isomap). Our second real-time video texture generation approach applies IST-Isomap and AR process and can produce better quality results for sparse videos such as cartoons. Based on our experiments, both of our methods can process the image frame data sequentially and more efficiently in terms of computational efficiency and time-consuming than the batch methods.

The rest of this paper is organized as follows: The original Isomap approach is briefly introduced in section 4.2. The incremental version of Isomap is discussed in section 4.3. Section 4.4 explains the IST-Isomap algorithm. Experimental results for generating real-time video textures are demonstrated in section 4.5.

4.2 Isomap

Isometric feature mapping (Isomap) [27] is a global nonlinear dimensionality reduction technique which reduces the dimensionality by searching a low dimensional manifold hidden in observational space. If we have a data set $\{X\}$ which contains n data points: $(\mathbf{x}_1, ..., \mathbf{x}_n)$ in a D-dimensional space, Isomap can be applied to map $\mathbf{x}_i$ onto a lower dimensional space with the representation $\mathbf{y}_i$ by preserving the geodesic distances between data points in the original space. The geodesic distance between two data points is defined as the shortest curve on the manifold connecting the two data points.

Isomap algorithm contains three major steps. The first step is to determine the neighborhood relations for all data points by constructing a weighted graph. We choose k-nearest neighbors ($k$nn) algorithm to represent neighborhood relations. For any two data points $\mathbf{x}_i$ and $\mathbf{x}_j$, $\mathbf{x}_j$ is considered as a neighbor of $\mathbf{x}_i$ if $\mathbf{x}_j$ is one of the k nearest neighbors of $\mathbf{x}_i$. The Euclidean distance between these two vertices is defined as Δ_{ij}. The value of k is manually defined. After obtaining the

relationships for all data points, a weighted undirected neighborhood graph G can be constructed by defining each data point as a vertex in this graph. The edge e_{ij} between two vertices v_i and v_j exists only if these two data points $\mathbf{x}_i$ and $\mathbf{x}_j$ are neighbors for each other. The weights of the edges are set to Euclidean distance between two vertices.

The next step is to discover the geodesic distances between all pairs of data points on the manifold. Here, the assumption is that we can estimate the geodesic distance between two data points approximately by the shortest path between the corresponding vertices in the neighborhood graph. We define g_{ij} as the length of the shortest path $sp(i, j)$ between two vertices v_i and v_j.

The last step is to apply classical multidimensional scaling (MDS) [49] to discover $\mathbf{y}_i$ in a lower-dimensional space d. We define the data points in the lower dimensional space as $\{\mathbf{Y}\} = (\mathbf{y}_1, ..., \mathbf{y}_n)$. The matrix $\mathbf{B}$ is the target inner products between different x_i, it can be calculated as $\mathbf{B} = -\mathbf{HSH}/2$, where $\mathbf{S}$ is the matrix of squared distances $\{s_{ij} = g_{ij}^2\}$ and $\mathbf{H} = \{h_{ij}\}$ is the "centering matrix" $\{h_{ij} = \delta_{ij} - 1\}$, δ_{ij} is the Kronecker delta. Since $\mathbf{Y}^T\mathbf{Y}$ is required to be set as close to $\mathbf{B}$ as possible, this can be done by $\mathbf{Y} = (\sqrt{\lambda_1}\mathbf{v}_1, ..., \sqrt{\lambda_d}\mathbf{v}_d)$, where $\lambda_1, ..., \lambda_d$ are the d largest eigenvalues of the matrix $\mathbf{B}$, and $\mathbf{v}_1, ..., \mathbf{v}_d$ are corresponding eigenvectors.

4.3 Incremental Isomap

Incremental Isomap was proposed by Law *et al.* [47], is a modified version of the original Isomap. First assume that we have already obtained the low-dimensional coordinates $\mathbf{y}_i$ of $\mathbf{x}_i$ for the first n data points. Then, a new data point $\mathbf{x}_{n+1}$ is observed in the sequence of data. Therefore, we need to update the existing low-dimensional coordinate set of $\mathbf{y}_i$ and estimate $\mathbf{y}_{n+1}$. Similar to the original Isomap algorithm, there are three steps for the incremental Isomap. The first step is to update the neighborhood graph and then update the geodesic distance g_{ij} in the view of the change of the neighborhood graph when the new data point $\mathbf{x}_{n+1}$ is observed. The second step is to apply the geodesic distance of the new point to the existing points to estimate the $\mathbf{x}_{n+1}$'s low-dimension coordinate $\mathbf{y}_{n+1}$. The last step is to update all low-dimensional coordinates set $\mathbf{x}_i$ after obtaining

the new geodesic distance g_{ij}.

4.3.1 Updating the Neighborhood Graph and Geodesic Distance

After observing the new input data point $\mathbf{x}_{n+1}$, the new vertex v_{n+1} is inserted into the neighborhood graph. We define A and D as the set of edges needed to be added and deleted after inserting the new vertex, respectively. For any vertex v_i, we set τ_i as the index of the kth nearest neighbor of v_i. A vertex v_j is the $k$nn neighborhood of v_i only if $\Delta_{ij} \leq \Delta_{i,\tau_i}$. Thus, the neighborhood graph is updated as following:

$A = \{e(i, n+1) : \Delta_{i,\tau_i} > \Delta_{i,n+1}\}$

$D = \{e(i, \tau_i) : \Delta_{i,\tau_i} > \Delta_{i,n+1}$ and $\Delta_{\tau_i,i}\Delta_{\tau_i,\iota_i}\}$

where ι_i is the index of the kth nearest neighbor of v_τ after the new vertex v_{n+1} is inserted into the graph. Then, the geodesic distance between v_{n+1} and the other vertices can be updated as follows:

$$g_{n+1,i} = g_{i,n+1} = \min_j(g_{ij} + w_{j,n+1}) \ . \tag{1}$$

4.3.2 Finding the Low-dimensional Coordinates of the New data

We can discover the low-dimensional coordinate $\mathbf{y}_{n+1}$ by matching its inner product with y_i to the values obtained from the geodesic distances. This is similar to the original Isomap approach. First, for the original Isomap, let's define:

$$g_{ij} = \| \mathbf{y}_i - \mathbf{y}_j \|^2 = \| \mathbf{y}_i \|^2 + \| \mathbf{y}_j \|^2 - 2\mathbf{y}_i^T\mathbf{y}_j \tag{2}$$

where $\sum_{i=1}^{2} \mathbf{y}_i = 0$. If we sum over j and then over i for g_{ij}, we can obtain

$$\| \mathbf{y}_i \|^2 = \frac{1}{n}(\sum_j g_{ij} - \sum_j \| \mathbf{y}_j \|^2) \tag{3}$$

and

$$\sum_j \| \mathbf{y}_j \|^2 = \frac{1}{2n} \sum_{ij} g_{ij} \,. \tag{4}$$

Similarly, in incremental Isomap, if we define $g_i = \| \mathbf{y}_i - \mathbf{y}_{n+1} \|$, then we can have

$$\| \mathbf{y}_{n+1} \|^2 = \frac{1}{n} (\sum_{i=1}^{n} g_i - \sum_{i=1}^{n} \| \mathbf{y}_i \|^2) \tag{5}$$

and

$$\mathbf{y}_{n+1}^T \mathbf{y}_i = -\frac{1}{2} (g_i - \| \mathbf{y}_{n+1} \|^2 - \| \mathbf{y}_i \|^2) \,. \tag{6}$$

Then, we can estimate the target inner product f_i between $\mathbf{y}_{n+1}$ and $\mathbf{y}_i$ as

$$2f_i \simeq \frac{\sum_j g_{ij}^2}{n} - \frac{\sum_{lj} g_{lj}^2}{n^2} + \frac{\sum_l g_{l,n+1}^2}{n} - g_{i,n+1}^2 \,. \tag{7}$$

Thus, we can obtain y_{n+1} by solving $\mathbf{Y}^T \mathbf{y}_{n+1} = \mathbf{f}$ in the sense of least square, where $\mathbf{f} = (f_1, ..., f_n)^T$. We can represent $\mathbf{Y} = (\sqrt{\lambda_1}\mathbf{v}_1, ..., \sqrt{\lambda_1 d}\mathbf{v}_d)^T$, here (λ_i, v_i) is an eigenpair of the target inner product matrix. This leat square problem can be solved as

$$\mathbf{y}_{n+1} = (\frac{1}{\sqrt{\lambda_1}} \mathbf{v}_1^T \mathbf{f}, ..., \frac{1}{\sqrt{\lambda_d}} \mathbf{v}_d^T \mathbf{f})^T \,. \tag{8}$$

The next objective is to update the whole set of low-dimensional coordinates in the view of the modified geodesic distance matrix. This can be observed as an incremental eigenvalue problem and the solution can be obtained by eigen-decomposition. The detailed algorithm is described in [47].

4.4 Incremental Spatio-temporal Isomap

Incremental Isomap works well for uncovering low-dimensional intrinsic data structure where data are collected sequentially. However, it is incapable to consider the temporal coherence in data. Spatio-temporal Isomap (ST-Isomap) [48] is an extension of the original Isomap framework which includes temporal relationships in data set. Nevertheless, it works in a batch mode and is not

appropriate when data are observed sequentially. We propose a new method named as incremental spatio-temporal Isomap (IST-Isomap), which is able to account for temporal coherence as while operate in a real-time manner. IST-Isomap inherits the framework of ST-Isomap and augments it with the ability to deal with sequentially collected data efficiently.

4.4.1 Spatio-temporal Isomap

Similar to the framework of original Isomap, the algorithm for ST-Isomap [48] contains five steps. The first step is windowing of the input data into temporal blocks in order to give a temporal history for each individual data point. This windowing step results in a sequentially ordered set of data points.

The second step is to identify the neighborhood relationships and build the neighborhood graph. The neighborhood graph in ST-Isomap is constructed based on common temporal neighbors (CTN) relationships. CTNs are determined by K-nearest nontrivial neighbors (KNTN) using Euclidean distance. A vertex v_a is considered to be a nontrivial match of v_b if $a = b + 1$, or $a \neq b$ and $\Delta_{b,a} \leq \Delta_{b,\tau_b}$ where τ_b is the index of the kth most similar nontrivial neighbor of v_b. We use D denotes the Euclidean distance matrix of the data set with each element Δ_{ij} represents the distance between two points i and j.

In the third step, the distance matrix D between data points is reduced with spatio-temporal correspondences. Other than CTN, another temporal relationship between data is defined as adjacent temporal neighbors (ATN). ATNs represent the adjacent points in the sequential order of data set. The reduced new distance $\Delta'_{i,j}$ between vertices v_i and v_j is calculated as following:

$$\Delta'_{ij} = \begin{cases} \Delta_{i,j}/(c_{CTN}c_{ATN}), & \text{if } v_j \in \text{CTN of } v_i \text{ and } v_j \in \text{ATN of } v_i; \\ \Delta_{i,j}/c_{CTN}, & \text{if } v_j \in \text{CTN of } v_i; \\ \Delta_{i,j}/c_{ATN}, & \text{if } v_j \in \text{ATN of } v_i; \\ \Delta_{i,j}, & \text{otherwise.} \end{cases} \tag{9}$$

where c_{CTN} and c_{ATN} are constant factors for controlling how strong the temporal relationships

between data pairs are. As c_{CTN} or c_{ATN} increases, the distance between data pairs with spatio-temporal correspondences decreases and their similarity increases.

In the fourth step, geodesic distance is calculated by applying Dijkstra's algorithm.

The final step applies classical MDS to construct an embedding of the data in low-dimension space as in original Isomap.

4.4.2 Incremental Version of Spatio-temporal Isomap

IST-Isomap inherits the framework of incremental Isomap and ST-Isomap. Assume that we already calculated the low-dimensional coordinates for the first n data points, then we need to update the low-dimensional coordinates for the existing data set after observing a new data point. In IST-Isomap, since we assume data are collected sequentially, it has the same result as the first step (windowing the input data) in ST-Isomap's algorithm. Therefore, the block size is one for data set in IST-Isomap which means each data point is treated as one single block as in ST-Isomap. The algorithm for IST-Isomap includes four steps:

1. Update the neighborhood graph. In IST-Isomap, whenever a new point $\mathbf{x}_{n+1}$ is observed, the existing neighborhood graph has to be updated corresponding to the new vertex v_{n+1}. Let τ_i be the index of the kth most similar nontrivial neighbor of v_i. It is straightforward to add a new edge from v_i to the new vertex v_{n+1} if v_{n+1} belongs to the CTN of v_i. The edge from v_i to v_{τ_i} is deleted if $\Delta_{i,\tau_i} > \Delta_{i,n+1}$ and $\Delta_{\tau_i,i} > \Delta_{\tau_i,\iota_i}$, where ι_i is the index of the kth most similar nontrivial neighbor of v_{τ_i} after inserting v_{n+1}. A new distance matrix D' for representing the neighborhood graph is obtained after this step.

2. Reduce the distance matrix D' between data points with spatio-temporal correspondences. This step is similar to the corresponding step in ST-Isomap, except that we only need to reduce the distance for vertex pairs where edges have been deleted or added.

3. Update geodesic distance matrix. Since the deleted and added edges may affect the geodesic distance of some vertices, the first task is to update these vertices with new distances followed by

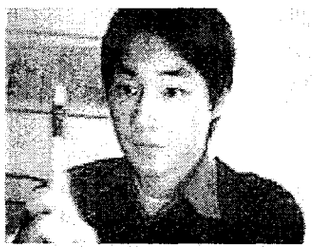


Figure 4.1: The input video.

calculating the geodesic distance between v_{n+1} and the other vertices by using (1).

4. In its final step, first the low-dimensional coordinates of the new data point $\mathbf{y}_{n+1}$ is calculated. Then the low-dimensional coordinates of the whole existing data set is updated with respect to the new geodesic distance. We apply the same algorithm for this step as in section 3.2 for incremental-Isomap.

4.5 Experimental Results

In our experiments, we implemented incremental Isomap and IST-Isomap on several input videos in order to produce real-time video textures. Our goal is to compare the quality and efficiency between the batch and incremental methods for generating video textures. Here, we apply same input videos as previous chapters. The only difference is now the frames are collected sequentially. For instance, the input video clip as shown in Figure 4.1 is 15 seconds long and contains 450 frames with a size of 160 $\times$ 128 (20480) pixels per frame.

4.5.1 Experimental Results by incremental Isomap

The preprocessing step is the initialization step. In this step, we are given the first 50 frames from an input video and then extract the signatures from these frames by applying the original Isomap. In our experiments, the dimensionality of 30 to 40 is good enough for representing each individual frame. This is done in the batch version of the Isomap, with a $k$nn neighborhood of size 7. After

(a) (b) (c)

Figure 4.2: The first three frames synthesized by incremental Isomap

this step, the rest of the frames in the input video will be observed one by one sequentially. Thus, the incremental Isomap is applied in order to extract the signatures of the up coming frames in a real-time fashion.

After preprocessing step, the rest of the process for generating new video textures contains three major steps: the first step is to implement incremental Isomap to extract the signatures when a new frame is coming from the input video. The next step is to apply an AR process to predict new frame signatures based on signatures obtained in the previous step[1]. Here, the order p of the AR process is set to 2, which means, the next frame signature will be generated based on the previous two signatures. Normally, in our experiments, we use the previous predicted frame signature and the new frame signature to predict the next frame. By combining the predicted signature with the new up coming signature, the level of noise in the result is reduced.

The last step is to project the new predicted frames signatures back to the image space and to compose them together as a video texture. Figure 4.2 shows the first three synthesized frames as a results of applying incremental Isomap to generate real-time video textures. Figure 4.3 shows the frames signatures resulted by incremental Isomap. We also tested our approach on some other video clips such as flame and fountain. The results are in Figure 4.4 and Figure 4.5.

Table 4.1 shows the comparison of runtime (seconds) for generating video textures between batch Isomap and incremental Isomap. Here, n is the number of new frames synthesized as the new video textures. "Dist" represents the time for distance computation for different n frames.

[1]Details of this step can be found in section 2.3

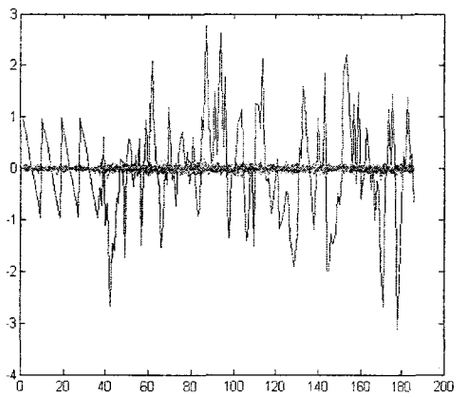
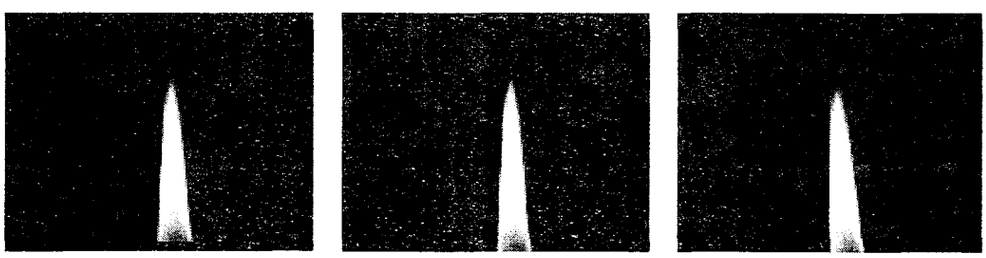


Figure 4.3: The frame signature synthesized by Incremental Isomap.

Table 4.1: Runtime (Seconds) for Generating Video textures by Original Batch Isomap and Incremental Isomap.

n	Dist.	Batch	Incremental	Speed-improved
50	0.19	10.45	3.78	63.83%
100	0.56	16.15	7.41	54.12%
150	0.83	21.54	11.35	47.31%
200	1.36	28.78	16.84	41.49%
250	1.88	36.12	20.09	44.38%



(a) (b) (c)

Figure 4.4: The first three frames synthesized by incremental Isomap

Table 4.2: Runtime (Seconds) for Generating Video textures by Original Batch Isomap and Incremental Isomap.

n	Dist.	Batch	Incremental	Speed-improved
50	0.08	9.03	2.55	71.76%
100	0.19	14.54	5.64	61.21%
150	0.43	19.30	9.21	52.28%
200	0.61	25.17	14.19	43.62%
250	0.98	32.89	18.03	45.18%

(a) (b) (c)

Figure 4.5: The first three frames synthesized by incremental Isomap

Table 4.3: Runtime (Seconds) for Generating Video textures by Original Batch Isomap and Incremental Isomap.

n	Dist.	Batch	Incremental	Speed-improved
50	0.17	11.59	4.50	61.17%
100	0.40	15.32	6.93	54.77%
150	0.79	18.77	13.38	28.72%
200	1.22	25.04	17.67	29.43%
250	1.63	31.28	21.10	32.54%

Table 4.4: Runtime (Seconds) for Generating Video textures by Original Batch Isomap and Incremental Isomap.

n	Dist.	Batch	Incremental	Incremental ST
50	0.24	13.89	6.19	7.62
100	0.68	18.24	10.22	10.03
150	1.05	24.17	14.09	13.79
200	1.51	34.51	20.78	18.47
250	1.97	40.56	26.42	24.61

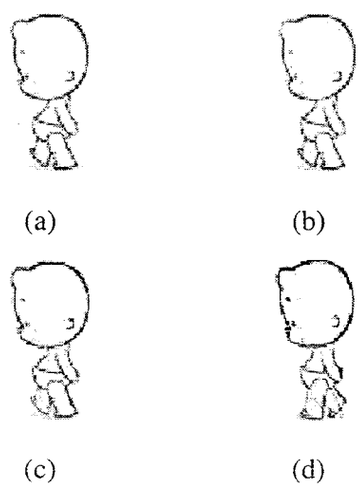


Figure 4.6: Compare the real-time video texture results generated by IST-Isomap and incremental Isomap for a cartoon animation: (a) and (b) are the 30th and 40th frames synthesized by IST-Isomap; (c) and (d) are the 30th and 40th frames synthesized by incremental Isomap

Tables 4.2 and 4.3 represent the comparison results for the other input video clips (flame and fountain, respectively). From these experimental results, we can conclude that, applying incremental Isomap to generate video textures is much faster than applying the original Isomap when the input data is available sequentially and the quality of the new video textures is appealing as while. Unproportionately, noise still exist, which in turn causes the frames to become blur after some period of time; however, in a short run, this provides us very good results.

4.5.2 Experimental Results by IST Isomap

In this section, we applied IST-Isomap to generate real-time video textures for cartoon data. In our experiments, we set the KNTN to 3, the values of constant factors c_{CTN} and c_{ATN} are set to 10 and 5, respectively (in our case, this provide the best results). The result is compared with the one generated by incremental Isomap as shown in Figure 4.6. It is clearly shown that the noise is noticeable in the 40th frame for incremental Isomap but not visible for IST-Isomap. Table 4.4 shows the comparison among batch Isomap, incremental Isomap and IST-Isomap in term of runtime (seconds) for generating video textures. According to the results, it is concluded that IST-Isomap is more robust to noise and more efficient for producing real-time video textures than incremental Isomap for cartoon data since they are inherently sparse and include exaggerated deformations between frames.

Chapter 5

Conclusions

Video texture is a new type medium which can provide a continuous, infinitely varying stream of video. It can synthesize an arbitrary length of video from a short input video clip. However, this technique can only generate video textures by just switching the order of frames in the input video and the results would suffer from 'dead-ends'. In this thesis, we have presented several new approaches for synthesizing video textures.

First, we have extended the application of PCA and AR process to generate video textures by replacing PCA with five other dimension reduction techniques (PPCA, Kernel PCA, Isomap, LLE and ICA). Based on our experiments, by using these dimensionality reduction techniques, the quality of video textures are improved compared with applying PCA to extract frame signatures. The synthesized video textures may contain similar motions as the input video and will never be repeated exactly. All the frames in the results are synthesized and have never appeared before. By comparing the experimental results among these dimensionality reduction techniques, PPCA provides more typical signatures which can be used to synthesize video texture than others. But unfortunately, noise may still appear and makes the video frame blur after some period of time.

Second, we have proposed a new approach for generating video textures using PPCA and GPDM. GPDM is a nonparametric model for learning high-dimensional nonlinear dynamical data sets. We have tested PPCA and GPDM on several movie clips, it can generate video textures containing frames that never appeared before with similar motions as the original video. Compared with PCA and AR process, PPCA and GPDM can produce better results with more robustness to

noise. Unfortunately, video textures synthesized by PPCA and GPDM still have visual discontinuities for some highly structured and variable motions (e.g. dancing and fighting). Thus, there might be some more potential improvements on generating video textures. Since GPDM is highly dependent on the kernel functions, selection of a better function would be a key factor for improving the predictive power.

Last, we have proposed two approaches of generating real-time video textures by applying the incremental Isomap and IST-Isomap. Both approaches can produce good real-time video texture results. In particular, IST-Isomap is more suitable for sparse video data (e.g. cartoon). Generating real-time video textures is extremely useful for some scenarios where data points are available sequentially (e.g. surveillance). Based on our experimental results, the runtime for creating new video textures is much faster with incremental Isomap and IST-Isomap than with the original Isomap in batch mode. The results are not only visually appealing with similar motions as the original video, but also contain frames that have never appeared before.

In conclusion, compared to the original video texture technique, our approaches not only can generate video textures with new frames that have never appeared in the input video, but problems of "dead ends" are also prevented. Our approaches can also process sequential data more efficiently than the original video texture technique, this can be considered as real-time video texture generation. Unfortunately, noise may still emerge in a long run and make the synthesized frame blur. Thus, one of our future works is to build a more robust statistical model in order to generate video textures with less noise but still in a real-time fashion.

List of References

[1] M. M. Oliveira. Image-Based Modeling and Rendering Techniques: A Survey. *RITA*, 9(2):37–66, 2002.

[2] A. Watt and F. Policarpo. *3D GAMES: Real-time Rendering and Software Technology*, volume One. ADDISON-WESLEY, 1 edition, 2001.

[3] H. Yeung, Shum and S. Kang. A Review of image-based rendering techniques. *Proc.SPIE*, 4067:2–13, May 2002.

[4] J. Foley. Getting There: The Ten Top Problems Left. *IEEE Computer Graphics and Application*, 20(1):66–68, 2000.

[5] A. Schödl, R. Szeliski, D. Salesin and I. Essa. Video Textures. In *Proceedings of SIGGRAPH 2000*, pages 489–498, July 2000.

[6] C. Bregler, M. Covell and M. Slaney. Video rewrite: Driving visual speech with audio. *Computer Graphics (SIGGRAPH97)*, pages 353–360, August 1997.

[7] L. P. Kaelbling, M. L. Littman and A. W. Moore. Reinforcement learning: A survey. In *Journal of Artificial Intelligence Research*, April 1996.

[8] A. Schödl and I. Essa. Controlled animation of video sprites. *In ACM SIGGRAPH Symposium on Computer Animation*, pages 121–128, July 2002.

[9] A. Schödl and I. Essa. Machine learning for video-based rendering. *In Advances in Neural Information Processing Systems.*, 13:1002–1008, 2001.

References

[10] S. Pollard, M. Pilu, S. Hayes and A. Lorusso. View synthesis by trinocular edge matching and transfer. *In British MachineVision Conference (BMVC98)*, September 1998.

[11] A. Finkelstein, C. E. Jacobs and D. H. Salesin. Multiresolution video. In *Proceedings of SIGGRAPH 96*, pages 281–290, 1996.

[12] A. Witkin and M. Kass. Spacetime Constraints. In *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 159–168. ACM, August 1988.

[13] Y. Li, T. Wang and H. Shum. Motion Texture: A Two-Level Statistical Model for Character Motion Synthesis. In *Proceedings of ACM SIGGRAPH 2002*, pages 465–472, New York, NY, USA, 2002. ACM.

[14] B. Celly and V. Zordan. Animated People Textures. In *Proceedings of the 17 th International Conference on Computer Animation and Social Agents*, 2004.

[15] L. Kovar, M. Gleicher and F. Pighin. Motion graphs. *ACM Transactions on Graphics*, 21(3):473–482, July 2002.

[16] O. Arikan and D. A. Forsyth. Interactive Motion Generation from Examples. *ACM Transactions on Graphics*, 21(3):483–490, July 2002.

[17] J. Lee, J. Chai, P. Reitsma, J. Hodgins and N. Pollard. Interactive control of avatars animated with human motion data. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 491–500, San Antonio, Texas, USA, 2002. ACM.

[18] N. Campbell, C. Dalton, D. Gibson and B. Thomas. Practical generation of video textures using the auto-regressive process. *Image and Vision Computing*, 22(10):819–827, September 2004.

References

[19] R. Kalnins, L. Markosian, B. Meier, M. Kowalski, J. Lee, P. Davidson, M. Webb, J. Hughes and A. Finkelstein. WYSIWYG NPR: drawing strokes directly on 3D models. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 755–762, 2002.

[20] V. Kwatra, A. Schödl, I. Essa, Greg Turk and A. Bobick. Graphcut textures: Image and video synthesis using graph cuts. *ACM Transactions on Graphics*, 22(3):277–286, July 2003.

[21] A. Agarwala, C. Zheng, C. Pal, M. Agrawala, M. Cohen, B. Curless, D. Salesin and R. Szeliski. Panoramic Video Textures. *ACM Transactions on Graphics*, 24(3):821–827, August 2005.

[22] S. M. Pandit and S. M. Wu. *Time series and system analysis with applications*. John Wiley & Sons Inc., 1983.

[23] A. W. Fitzgibbon. Stochastic rigidity: Image registration for nowhere-static scenes. In *Proceedings Computer Vision, 2001. ICCV 2001. Eighth IEEE International Conference*, pages 662–669, 2001.

[24] Christopher M. Bishop. Probabilistic Principal Component Analysis. In *Journal of the Royal Statistical Society, Series B*, pages 611–622, 1999.

[25] C. M. Bishop. Mixtures of probabilistic principal component analyzers. *Neural Computation*, pages 443–482, 2000.

[26] A. Smola and K. Mller. Nonlinear Component Analysis as a Kernel Eigenvalue Problem. pages 1299–1319. Neural Computation, 1998.

[27] J. B. Tenenbaum, V. de Silva and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. pages 2319–2323. Science 290, 2000.

[28] E. W. Dijkstra. A note on two problems in connexion with graphs. pages 269–271. Numerische Mathemati, 1959.

[29] R. W. Floyd. Algorithm 97: Shortest path. page 345. Communications of the ACM, 1962.

[30] O. C. Jenkins and M. J. Mataric. A spatio-temporal extension to Isomap nonlinear dimension reduction. In *Proceedings of the twenty-first international conference on Machine learning*, pages 441–448. ACM, 2004.

[31] S. Roweis and L. Saul. Nonlinear dimensionality reduction by locally linear embedding. pages 2323–2326. Science 290, 2000.

[32] A. Hyvarinen and E. Oja. Independent component analysis: algorithms and applications. Neural Networks, 2000.

[33] C. Juan and B. Bodenheimer. Cartoon textures. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 267–276, 2004.

[34] J. Y. Dong, C. X. Zhang, Y. J. Wang and H. Y. Zhou. A video texture synthesis method based on wavelet transform. In *Wavelet Analysis and Pattern Recognition, 2007. ICWAPR '07. International Conference*, volume 3, pages 1177–1181, Nov 2007.

[35] C. E. Rasmussen and C. Williams. Gaussian processes for machine learning. the MIT Press, 2006.

[36] D. J. C. MacKay. Information Theory, Inference, and Learning Algorithms . Cambridge University Press, 2003.

[37] D. J. C. MacKay. Introduction to Gaussian processes. In *Neural Networks and Machine Learning*, volume 168, pages 133–165. Springer, 1998.

[38] J. M. Wang and D. J. Fleet and A. Hertzmann. Gaussian process dynamical models. In *Advances in Neural Information Processing Systems 18*, pages 1441–1448. MIT Press, 2006. Proc. NIPS'05.

[39] J. M. Wang and D. J. Fleet and A. Hertzmann. Gaussian process dynamical models for human motion. In *Pattern Analysis and Machine Intelligence*, pages 283–298. IEEE Transactions, Feb 2008.

[40] N. Lawrence. Probabilistic non-linear principal component analysis with Gaussian process latent variable models. In *The Journal of Machine Learning Research*, pages 1783 – 1816. MIT Press, Cambridge, MA, USA, 2005.

[41] R. Shumway and D. Stoffer. Time Series Analysis and its Applications. Springer, 2000.

[42] W. Fan and N. Bouguila. On video textures generation: a comparison between different dimensionality reduction techniques. In *IEEE International Conference on System, Man, and Cybernetics*, 2009.

[43] C. Rasmussen and C. Williams. Gaussian Processes for Machine Learning. The MIT Press, 2006.

[44] G. Doretto, A. Chiuso, Y. N. Wu and S. Soatto. Dynamic Textures. *International Journal of Computer Vision.*, pages 91–109, 2 2003.

[45] P. Saisan, G. Doretto, Y. N. Wu and S. Soatto. Dynamic Textures Recognition. *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on.*, page 58, 2001.

[46] G. Doretto, D. Cremers, P. Favaro and S. Soatto. Dynamic Texture Segmentation. *In IEEE International Conference on Computer Vision.*, pages 1236–1242, 2003.

[47] H. C. Law and A. K. Jain. Incremental Nonlinear Dimensionality Reduction by Manifold Learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence.*, 28(3):377–391, 2006.

[48] O. C. Jenkins and M. J. Mataric. A spatio-temporal extension to Isomap nonlinear dimension reduction. In *Proceedings of the twenty-first international conference on Machine learning*, pages 441–448. ACM, 2004.

[49] T. F. Cox and M. A. A. Cox. Multidimensional scaling. *Chapman and Hall*, 2001.