Perception-driven procedural texture generation from examples

Jun Liu[a], Yanhai Gan[b], Junyu Dong[c,*], Lin Qi[c], Xin Sun[c], Muwei Jian[d], Lina Wang[e], Hui Yu[f]

[a]*Science and Information College, Qingdao Agricultural University, 700 Changcheng Road, Qingdao, China*
[b]*Hisense TransTech Co., Ltd, China, No.17 Donghai West Road, Qingdao, China*
[c]*Department of Computer Science and Technology, Ocean University of China, 238 Songling Road, Qingdao, China*
[d]*School of Computer Science and Technology, Shandong University of Finance and Economics, Jinan, China*
[e]*China Unicom Institute of Software, N0.87 Huaneng Road, Jinan, China*
[f]*Department of School of Creative Technologies,University of Portsmouth, Eldon Building, Winston Churchill Avenue, Portsmouth PO1 2DJ, UK*

Abstract

Procedural textures are widely used in computer games and animations for efficiently rendering natural scenes. They are generated using mathematical functions, and users need to tune the model parameters to produce desired texture. However, unless one has a good knowledge of these procedural models, it is difficult to predict which model can produce what types of textures. This paper proposes a framework for generating new procedural textures from examples. The new texture can have the same perceptual attributes as those of the input example or re-defined by the users. To achieve this goal, we first introduce a PCA-based Convolutional Network (PCN) to effectively learn texture features. These PCN features can be used to accurately predict the perceptual scales of the input example and a procedural model that can generate the input. Perceptual scales of the input can be redefined by users and further mapped to a point in the perceptual texture space, which has been established in advance by using a training dataset. Finally, we determine the parameters of the procedural generation model by performing perceptual similarity measurement in the perceptual texture space. Extensive experiments show that our method has produced promising results.

Keywords: Procedural texture, Texture generation, Texture perception, Convolutional neural network, Deep learning, PCA-based convolutional network

*Corresponding author
Email address: dongjunyu@ouc.edu.cn (Junyu Dong)

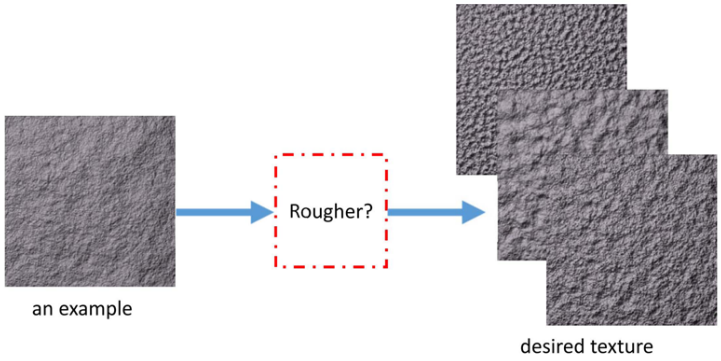

Fig. 1: The procedure of designing a "new" texture based on an example. The rock surface on the left is the example, and the surfaces on the right are the outputs that are a visually similar but look rougher than the example.

1. Introduction

Procedural textures have been widely used in computer games, animation and many other graphics applications for efficient rendering of natural elements, such as wood, marble, rocks, clouds and other materials[1]. They are typically created by procedural models, which are essentially mathematical functions and implemented using computer algorithms. The advantage of using procedural textures is that they require little storage and computation, and can be generated in real time. However, tuning the parameters of procedural models to produce desired textures is a difficult task even for experienced users. Unless one has a good knowledge of procedural texture models, it is difficult to predict which model can produce what types of textures. In addition, parameters of one model will produce overlapping effects on the output texture appearance. Therefore, it is hard to evaluate the influence of each parameter on the output texture.

For artists, designing a "new" texture that can be used in games or animations normally starts with an example of texture, e.g. a rock surface image downloaded from the internet or generated using procedural models. However, the example texture might not meet user expectations; changes in one or more of its perceptual properties are often required (e.g. the artist might wish the rock surface to look rougher). Fig. 1 illustrates the designing process. Currently, to our best knowledge, no software can provide direct solutions to this user requirement. Most packages only provide functions for manual editing, which is complicated and time-consuming. Even though some powerful texture generators (e.g. Genetica, FilterForge) can create high-quality textures or animated textures from given images, they are not able to make direct modifications to perceptual attributes, e.g. modifying roughness, directionality or regularity of the input texture. It is also the case for finding proper procedural textures, i.e. a "new" procedural texture whose perceptual characteristics are different

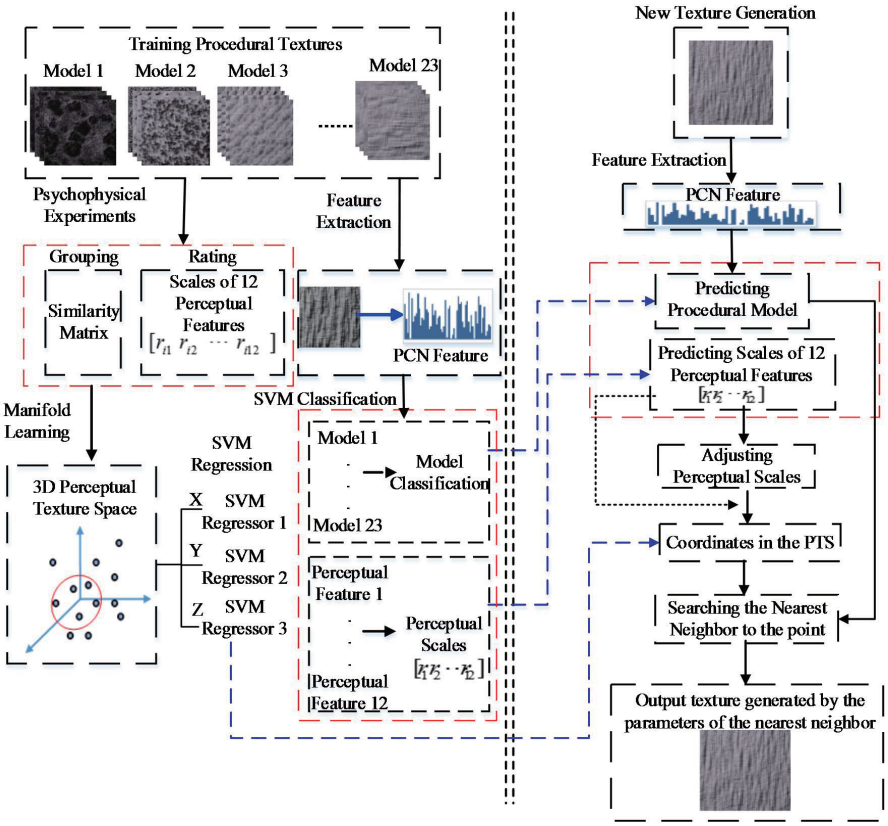

Fig. 2: The framework of the proposed approach. The left part shows the training process and the right part shows the process for generating a new texture.

from the example. This is indeed an even more difficult task, because generation of a procedural texture with different perceptual properties involves finding both proper models and parameter settings. With only an example procedural texture as input, current commercial software cannot determine its generation models and corresponding parameters.

In this paper, we propose a novel approach for generating a new texture with different perceptual properties yet sharing certain similarity to the example image. Fig. 2 shows the framework. The input to the system is an example texture, and the system can automatically find a procedural model and determine the parameters to output a new texture. In the proposed approach, users are allowed to adjust one or more perceptual properties of the input texture. Thus, the new texture bears resemblance to the example, while certain perceptual features can be perceived differently from the example. The left block of Fig. 2 shows the training process. First, we use a training dataset introduced

in the previous work [2]; the procedural textures in this data set are generated by 23 procedural models and annotated with 9-point Likert scales for twelve perceptual features. We call these annotated values as perceptual scales in this paper, assessing to what extent the features are perceived by subjects. The similarity matrix derived from the grouping experiment is used to construct the perceptual texture space (PTS) [2], while the perceptual scales of the training samples from the rating experiment are used to train regression models.

The right part of Fig.2 shows the process for generating a new texture. For an input texture, computational features are extracted using deep networks and the perceptual scales are predicted by employing a pre-trained SVM model. Meanwhile, the procedural model that can generate the example is also predicted. Then, the perceptual scales of the input texture can be mapped to a point in the PTS based on regressing the coordinates of the PTS. They can also be adjusted by the user and further mapped to a point in the PTS. Next, we perform similarity measurement in the PTS by finding the nearest neighbor to the point representing the texture we wish to generate. Since each point in the PTS represents a texture with known procedural models and corresponding parameters, we can determine the model and corresponding parameters to generate the new texture similar to the input or with desired perceptual scales.

The main contributions of this paper are twofold. First, we propose a method for generating procedural textures that are visually similar to the example or with user-defined perceptual properties. The proposed scheme supports a variety of procedural models. In addition, the perceptual features of the texture can be determined accurately, which are consistent with human perception. Second, a PCA-based Convolutional Network (PCN) is proposed for texture feature extraction. We would like to design a simple yet effective deep learning method. Ideally the network should be simple for training and able to learn features that can adapt to different datasets, even a dataset with a small number of samples. Compared to existing deep convolutional networks, the proposed PCN is composed of unsupervised pre-training stages. It does not involve regularized parameters and does not require numerical optimization solver either; these make the training procedural more efficient and require less time to obtain PCA filters. Moreover, it can learn effective features even with a small training dataset. The PCN also achieved state-of-the-art performance on several tasks based on publicly available datasets, including hand-written digital recognition, face recognition and texture classification.

The rest of the paper is organized as follows. Section 2 reviews relevant research to our work. Section 3 introduces the PCA-based Convolutional Network (PCN) for extracting texture features. Section 4 presents details of the proposed framework. Section 5 reports our experimental results. Finally, we conclude the paper in Section 6.

2. Related work

In this section, we briefly review works on generating procedural textures from examples. We also list recent works on texture feature extraction, including

deep learning methods.

2.1. Procedural texture generation

Procedural texture models are widely used in many research and application fields such as efficiently adding rich detail to synthetic scenes [1, 3]. A set of input parameters to mathematical models controls the output textures that are perceived differently by human subjects. For example, textures with different perceived roughness can be generated by controlling parameters in the fractal models [4]. There are also few attempts to generate desired noise texture by controlling the parameters of procedural models in the frequency domain [5, 6] and spatial domain [7, 8]. Other works in controlling parameters include extracting noise functions from an example image, so that a new noise model can be designed automatically to produce an image closely resembling the example [9, 10, 11, 12, 13, 14, 15]. However, these solutions are only suitable for a particular type of procedural noise model. Gilet et al. propose a method to determine parameters for multi-scale stochastic functions by using an error minimization technique. This technique calculates the image distance based on local filter banks; and their method can be applied to random textures[16]. A more robust method is proposed to automatically choose parameters for procedural models based on an example[17]. The set of parameters are retrieved from a pre-computed database by applying the texture distance metric. Although the process needs pre-selection of a model type, their approach is generalized for different types of procedural texture models. Our approach is related to these works; the novelty is that our method can automatically find procedural models and corresponding parameters to not only generate the input example, but also produce new textures with user-defined perceptual attributes.

2.2. Texture features

In the past decades, great efforts have been made to design hand-crafted features for different tasks, such as classification, retrieval and discrimination[18]. The most popular types of such kind of features include Gabor wavelets[19], Local Binary Patterns (LBP)[20], HOG[21] and SIFT[22]. However, these features are outperformed by those based on deep learning in many publicly available data sets.

Deep learning is becoming a popular way for automatically learning features that disentangle the underlying factors of variations. Convolutional Neural Network (ConvNet) is one of the most powerful deep architectures for learning features[23]. In recent years, CNN has been used for image and also texture generation[24, 25, 26].

A deep ConvNets with multistage architectures can learn hierarchical features, from low-level to high-level features. However, training such a deep network typically uses a gradient descent method in a supervised mode, which always needs large amounts of labelled samples for training. In addition, good results sometimes depend on the tricks of the trade for parameters tuning.

Variations of ConvNets have been proposed with respect to the pooling and convolutional layers. The typical system uses unsupervised pre-training in each

layer and global supervised training to fine-tune the whole system. A variety of techniques were proposed to pretrain filter banks in convolutional layers. The convolutional versions of sparse RBMs [27], sparse coding [28] and predictive sparse decomposition (PSD) [27, 29, 30] were reported and achieved high accuracy on several benchmarks. Alternatively, the networks similar to ConvNets were proposed by using pre-fixed filters in convolutional layers. Gabor filters were used in the first convolutional layer[31, 32]. Wavelet scattering networks (ScatNet) [28, 33] also used pre-fixed convolutional filters which were called scattering operators. By using a similar multiple levels of ConvNets, the algorithm had achieved impressive results in handwritten digits and texture recognition.

More recently, Chan et.al proposed a network, called PCANet, of which convolutional filter banks in each stage are simply PCA filters[34]. Built upon a layer-wise convolutional layer, binary hashing and block-wise histograms are used in the last nonlinear stage and with a supervised classifier on the top. Surprisingly, such a network with just a few cascaded convolution layers has demonstrated competitive performance in several challenging vision tasks, such as face and handwritten recognition, and also comparative results on texture classification and object recognition. It is especially suitable for those tasks when only a few labeled samples are available.

Considering the fact that procedural textures are different from real-world photographs, we propose an improved method for texture feature extraction based on PCANet and CNN; we name the new method as the PCA-based Convolutional Network (PCN). The network has both advantages from PCANet and CNN, and can effectively extract texture features for predicting perceptual scales.

Our work is also related to recent work on texture description and recognition with semantic attributes. Cimpoi et.al [35, 36, 37] collected the describable texture dataset (DTD) that was annotated with 47 perceptual texture attributes. They ported Fisher Vector (FV) as a pooling method to texture domain; the combination of IFV and DeCAF (IFV+DeCAf) or CNN (FV-CNN) outperformed the state-of-the-art texture representations in recognizing material and texture attributes. Recently, Lin et.al [38] studied the bilinear CNN features for texture recognition and visualized inverse images for various categories learned by these models. They applied the approach for manipulating images with semantic attributes. It provided a unified parametric model of texture representation and recognition. Differing from these works, we propose a novel approach to generate procedural textures with perceptual features based on example texture. The perceptual features can be adjusted, e.g. to make a texture perceived more regular or uniform, and the generation model and corresponding parameters can be automatically decided. To our knowledge, this is an important issue and has not been investigated in previous work.

3. The PCA-based convolutional networks for texture feature extraction

In our framework, texture features are important for predicting perceptual scales and generation models. We will first describe the PCA-based Convolutional Networks (PCN) that can effectively extract texture features. PCANet is a simple deep learning network, and can effectively extract useful information for different tasks. However, the dimension of the resulted feature would increase exponentially with the number of stages (layers). That limits PCANet to grow deeper and the performance. The idea of PCN is to fix the problem of PCANet and make the features more effective. Like the PCANet, the PCN is composed of unsupervised pre-training stages and a nonlinear output stage. Generally, there are two differences of PCN with PCANet. First, we add a pooling layer right after the convolutional layer in each feature extraction stage. The pooling operation results in feature maps with reduced resolution, and these pooling features are translation invariant. Second, we group the multiple sets of feature maps into subsets according to certain rules. The PCA filters in the next convolutional layer are trained based on the subsets of feature maps. This may be seen as weight sharing of receptive fields in ConvNets. Feature maps in one subset capture certain features of the input images, whereas those in different subsets capture different types of features. These strategies accelerate the training process and the output features can capture more effective representation of the image.

The structure of a typical PCN contains three stages. Each pre-training stage consists of a convolutional layer and a pooling layer. The inputs are first convoluted with PCA filters to produce a set of feature maps, and then average or max pooling is conducted to aggregate feature maps into effective ones with small size. These feature maps are further combined by certain rules and fed into the next stage as input. In the output stage, we perform binary hashing and block histogram to produce the final output features.

3.1. The first convolutional stage

Suppose we are given N training samples which are denoted as $\{I_i\}_{i=1}^{N}$; the size of each input image is $m \times n$. The filter size used in each stage is represented as $k_1 \times k_2$.

3.1.1. Pre-processing

First, we take patches from the training samples. The patches are sampled every k pixels, and each patch is $k_1 \times k_2$ pixels(k should less than k_1, k_2 and the patches are overlapping). For each patch, it is scanned in raster order to form a vector with the patch mean removed. Then for all the training samples, mean-removed patch vectors are put together to form a matrix $X = [\bar{X}_1, \bar{X}_2, \bar{X}_3, \cdots, \bar{X}_N]$, where $\bar{X}_i$ is the matrix formed by mean-removed patches vectors of the i_{th} sample.

3.1.2. Filter banks

The filter banks are learned using Principle Component Analysis (PCA). We perform PCA on the matrix X. The PCA filters are selected as the first L_1 principle eigenvectors of XX^T. The eigenvectors are reshaped to the size $k_1 \times k_2$. In this way, we obtain L_1 filters with size $k_1 \times k_2$.

3.1.3. Convolution

Subsequently, images are convolved with the filter bank to generate filter responses. The filter responses are called feature maps as follows:

$$I_i^l = I_i * W_l^1, i = 1, 2, 3, \cdots, N \tag{1}$$

where: "$*$" represents the 2D convolution operation; I_i is padded with zeros before convolution.

3.1.4. Pooling

The convolution with each input image produces L_1 feature maps. Then the max pooling or average pooling is applied to the pooling regions. We use $\{S_i^l\}_{i=1}^N, l = 1, 2, 3, \cdots, L_1$ to represent the pooling result of the lth feature map of the ith input image. Since there are L_1 filters in the first stage, we obtain NL_1 feature maps in total.

3.1.5. Feature map combination

These NL_1 feature maps are divided into L_1 subsets. Each subset includes N feature maps which are produced by convoluting the input images with the same filter, and they are denoted as $S^l = \{S_i^l\}_{i=1}^N, l = 1, 2, 3, \cdots, L_1$.

Since high level features are the combinations and abstraction of low level features [23], we combine subsets $\{S^l\}_{l=1}^{L_1}$ according to certain rules to form several groups. For example, suppose there are 5 filters in the first stage which result in 5 subsets. Two of the subsets are added and then 5 new groups corresponding to five new patterns are formed. This is similar to the connections among feature maps of neighboring layers in CNN. In practice, an indexing matrix is used to define the way of combination. In the indexing matrix, most entries are zeros and a few entries are ones, which indicate the subsets belonging to one group (i.e. a new subset). The combination produces several new subsets and each new subset is denoted as $\{S_i^{l'}\}_{i=1}^N$, which also consists of N feature maps.

3.2. The second convolutional stage

The operations are mostly repeating the same procedure as in the first stage. For each $\{S_i^{l'}\}_{i=1}^N$, we sample patches from each feature map in this subset. Then the vectorized patches are mean-removed and put together to form a matrix denoted as $Y^{l'} = [\bar{Y}_1^{l'}, \bar{Y}_2^{l'}, \bar{Y}_3^{l'}, \cdots, \bar{Y}_N^{l'}]$. Because there are L_1 subsets, we obtain L_1 matrixes $Y^{l'}, l' = 1, 2, 3, \cdots, L_1$.

For each subset $\{S_i^{l'}\}_{i=1}^N$, we learn filters by performing PCA separately and choose the first L_2 leading principle eigenvectors as PCA filters, denoted as

$\{V_l^{l'}\}_{l=1}^{L_2}$. Each feature map in this subset is convoluted with L_2 filters; this results in L_2 new feature maps. Since there are L_1 subsets (produced by L_1 groups), $L_1 N L_2$ feature maps are obtained.

The pooling process in the second stage is as same as in the first stage. After pooling, the feature maps are considered as the output of the second stage.

3.3. Output stage

In the output stage, we simply use binary quantization and block histogram as in PCANet[34]. The binary quantization converts the outputs in the last stage back into a single integer-valued "image". Then we partition the integer-valued "image" into B blocks and compute the histogram in each block. Finally, we concatenate all the B histograms into one vector, which is used as the final output features.

The parameters of the PCN include the number of stages, the number of filters in each stage, the size of the filters (i.e. the patch size) k_i, the patch step k, the ways of feature map combination, the pooling size $p \times q$ and the block size for histograms. Typically, the two-stage PCN can achieve promising performance. The number of filters in each stage may be determined by validation for the best performance. The size of the filters (for two-stage PCN) k_1 and k_2 should satisfy the condition of $k_1 * k_2 >= L_1, L_2$. Also, the pooling size is typically set to 2 for each stage. The patch step and the block size should be determined according to the image size and image type. Moreover, we notice empirically that when the identity matrix is used as the way for feature maps combination, it is normally sufficient to achieve good performance.

4. Perception-driven texture generation

This section describes our method for generating procedural textures according to an example or user-defined perceptual scales. The method is based on the procedural texture dataset introduced in [2]. The dataset comprises 450 images generated by 23 procedural models[1]. Each texture has 512×512 pixels and is rendered under the same lighting conditions. Two forms of psychophysical data are also included in the dataset. One is the similarity s_{ij} between the textures t_i and t_j, which is evaluated by sorting textures into groups according to visual similarity. The other is the averaged twelve perceptual scales [39][2] r_i for each texture. These scales are obtained from the subjective rating experiment on the

[1]The 23 methods are CA (Forest fire model), CA (Surface tension model), CA (Excitable media model), Cellular, Folding of Texton Placement, Folding of Cellular, Folding of Fractal, Folding of Perlin noise, Fractal (one-over-fBata-noise), Fractal (Fourier spectral synthesis), Fusion of Cellular and Texton, Fusion of Perlin and Cellular, Fusion of Perlin and Texton, Islamic Patterns, Matrix Transformation, Perlin noise, Reaction Diffusion, Texton Addition, Texton probability map, Texton random grid, Texton random walk, Texton regular.

[2]The 12 perceptual features include: (in order) contrast, repetition, granularity, randomness, roughness, feature density, directionality, structural complexity, coarseness, regularity, local orientation, and uniformity.

9-point Likert scales. Each perceptual scale r_i is used for measuring the degree that certain visual property belong to a texture. The details of the dataset can be found in [2].

4.1. Predicting perceptual scales

This section discusses our method for automatically predicting the perceptual scales for the input example T_{input}; the predicted perceptual scales can well describe the example texture. In the training dataset, each texture has two representations: one is the PCN feature vector z_i and the other is perceptual scale vector $r_i = [r_{i1}\ r_{i2}\ \cdots\ r_{i12}]$. The scales for each perceptual feature can also be regarded as class labels (1 to 9). The prediction of perceptual scales for the input texture T_{input} is now transformed to classification. For a certain perceptual feature, we can classify the texture samples into 9 classes corresponding to their Likert scales. In our method, SVM is employed for classification. Since we use 12 perceptual features as in [2], twelve SVM classifiers $CModel_i = [CModel_1\ CModel_2\ \cdots\ CModel_{12}]$ are trained and each classifier is responsible for mapping the PCN features to one perceptual feature accordingly, that is for the input texture T_{input}, $r_{inputj} = CModel_j(z_{input})(j = 1, \cdots 12)$. Fig. 3 shows the block diagram of predicting perceptual scales of the 12 features.

4.2. Mapping the input texture into the perceptual texture space

This section introduces our method to map the input texture into the perceptual texture space (PTS). We map the input texture into the PTS by using the 12 perceptual scales r_{input} predicted in the last section. The reason we choose to map the perceptual scales into the PTS is twofold. First, the PTS is a feature space in which the dimensions can be used as a standard representation for texture and for perceptual similarity judgement. In [2], it has been shown that the similarity measure in PTS is more consistent with human perception. Second, if users want to generate a texture similar to the input example, but with changes of certain perceptual scales, they can easily modify the perceptual scales based on the predicted perceptual ones from the input example.

We construct the PTS the same as in [2]. The Isometric feature mapping (Isomap) algorithm [40] is applied to the similarity matrix to derive the PTS. The m dimensional embedding identifies the structure in the similarity information. Each point in the space represents a texture sample in the training

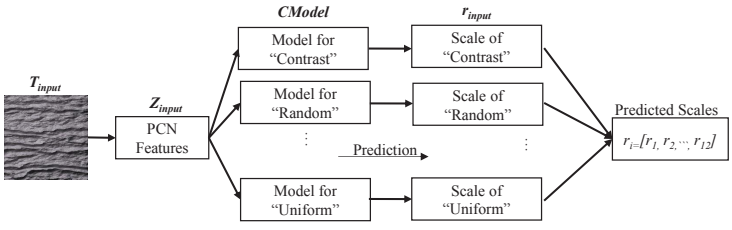

Fig. 3: The block diagram of predicting perceptual scales of the 12 perceptual features.

set and the Euclidean distances p_{ij} between points t_i and t_j in the perceptual space corresponds to the original similarity s_{ij} and perceptual distance. For our procedural texture dataset, three dimensions of the PTS are identified. The mapping process is shown in Fig. 4.

For the Isomap algorithm, there is no straightforward extension to out-of-sample examples. In terms of our method, it is impossible to conduct the free grouping experiment again as in [2] with a given out-of-sample example. We use regression method to find a set of functions $f_k(r)(k = 1, 2, 3)$ that map the perceptual scale vectors r of each texture to the coordinates p_k of PTS. That is $p_k = f_k(r)(k = 1, 2, 3)$. The regression is typically nonlinear and three SVM models $RModel_k(k = 1, 2, 3)$ are used to determine the nonlinear mapping. The underlying dimensions of the PTS are identified by computing correlations between each coordinate and 12 perceptual scales of the training samples. As demonstrated in [2], for each perceptual feature, we believe that the most relevant dimension of PTS is the one with the highest correlation coefficient to this feature. Thus, each dimension in the PTS are highly correlated with a certain number of perceptual features, instead of all the 12 features. In this way, for each perceptual feature, there is a correspondent dimension; accordingly, for each dimension, these corresponding features form a subset. Then, a subset of feature vectors is selected based on the correlation analysis to construct the mapping to each dimension in the PTS.

Altogether, three subsets of feature vectors $sr_i(i = 1, 2, 3)$ are formed as input training features to the SVM regression models.

In this way, any input perceptual scales can be mapped to a three dimensional vector corresponding to the coordinates of a point in the PTS.

4.3. Perception-driven procedural texture generation from examples

The goal of this stage is to find a procedural model and appropriate parameters to generate textures similar to the input example or with user-defined perceptual scales.

As the relationship between the textures and corresponding models in the training dataset is known, we are able to train a SVM classifier based on PCN features z_i with class labels $Label_n(n = 1, \cdots 23)$ representing different models. Thus, a procedural texture model can be firstly predicted for the input example

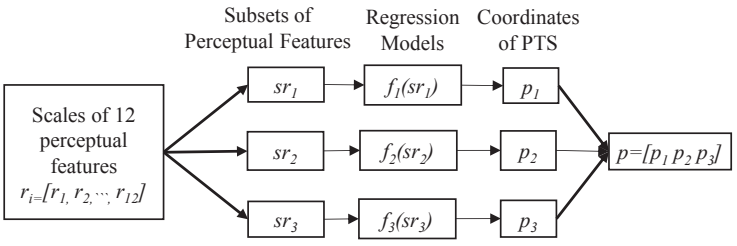

Fig. 4: Mapping from the perceptual features to coordinates in the PTS.

T_{input}. We use the predicted model to generate a set of textures, which are further mapped to points in the perceptual texture space (PTS). In general, varying model parameters in a model will only affect certain perceived texture features, while most of perceptual features remain unchanged [41]. We can therefore calculate texture similarity p_{ij} in the PTS to retrieve perceptually similar samples t_s within the set of points corresponding to textures generated by the relevant procedural model $Label_n$. Recall that in our dataset, for each model, we first generated a large number of textures by linearly increasing the value of model parameters. This ensures that the range of appearances of the generated texture can be sufficiently covered. Then, we selected samples with obviously different appearances from those produced by each model. We discretize and sample the parameter space on the linear scale for each parameter so as to produce textures perceived differently. The number of textures chosen from each model depended on the range of texture surfaces they could generate. The set of parameters for each sample is stored in the database.

Thus, since both the input example T_{input} or perceptual scales r_{input} and all generated texture are mapped to points in the PTS, by comparing the distance between points in the PTS, the model parameters $Para_n(t_s)$ can be assigned as those with the minimum distances i.e. the most similar texture t_s to the input. It is also the case if users specify perceptual scales as input. Although in the PTS, a manifold might not globally resemble the Euclidean space, each point has a neighborhood that is homeomorphic to the Euclidean space. Accordingly, the model parameters can be decided by the point that has the minimal Euclidean distance to the one corresponding to the input texture or changed perceptual scales (note that the modified perceptual scales are also mapped to a point in the PTS). With these parameters, we are capable of accurately producing new textures with user-defined perceptual scales, that is, $T_{new} = F(Label_n, Para_n(t_s))$, where F represents the mathematical function of the procedural model.

5. Experimental results and discussion

In this section, we first introduce the method to augment the training samples. Then we provide results on predicting procedural models as well as perceptual scales, and further we show the texture generation results based on perceptual features. Finally we evaluate the performance of the proposed PCN with other methods on publicly available datasets.

5.1. Data pre-processing and augmentation

We use the procedural texture dataset introduced in [2] for training the proposed method. Because more training data generally improves the performance of the deep networks, we also augment the samples using techniques of lighting variations [42]. This helps to prevent overfitting and helps the model generalizes better.

Each height map is rendered again under different lighting conditions to generate new texture images, but with the same illumination slant angle and tile

Table 1: Comparison of classification accuracy (%) on procedural textures dataset.

Methods	IFV+DeCAF	PCANet-2	PCN-2
Accuracy	96.70	99.62	**99.89**
Training Time(s)	446	16407	**251**
Test Time Per Sample(s)	0.31	3.14	**0.1136**

angle. This is because changes in illumination directions can cause significant variations in subjects' perception[43]. The second way use to increase the image number is to cut the training sample into 4 non-overlapping sub-images. As texture is constrained by properties of homogeneity or isotropy, we assume that the perceptual properties of the cropped images remain the same as the original ones. In total, 3600 textures are used as training data.

The perceptual similarity between original textures and new rendered textures with different lighting conditions can be evaluated using structural similarity index measurement (SSIM)[44]. The mean SSIM (mSSIM) index is used to evaluate the overall image quality. The highest and lowest scores are 0.9832 and 0.53 respectively. The mean score and standard variation for all the texture pairs are 0.8643 and 0.0749 respectively, which means that new rendered textures are perceptually similar to the original textures. In other words, the new textures preserve the same visual properties perceived by subjects. The perceptual scales of new textures may be assumed to be the same as the corresponding original one.

5.2. Results of predicting texture models and perceptual scales based on PCN

We first test the performance of PCN features on predicting procedural models and perceptual scales.

For textures generated by one model, we randomly select 25% images as the test sample, whereas the others are used as the training samples. We compare the PCN-2 with PCANet-2 and IFV+DeCAF, which was used in [35] and produced state-of-the-art performance on the Describable Texture Dataset (DTD). We set the parameters of the PCANet-2 as follows: the filter size is 7×7, the number of filters in both stage is 8 and the block size is 64×64. In the PCN-2, the patch sampling interval is set to 3, the filter size is set to 7×7, and the number of filters in each stage is set to 16 and 38 respectively. Maxpooling is used and the size is 2×2. The performance comparisons are shown in Table 1. Although the number of filters in each stage of PCN-2 is much more than that in PCANet-2, PCN-2 requires much less training time and achieves better performance compared to PCANet-2 and IFV+DeCAF. Thus, from the experimental results on our dataset, the proposed algorithm PCN is able to learn effective features on different kinds of databases, and achieve competitive accuracy, even on datasets with a small number of training samples.

We then feed the PCN features to SVM classifiers for predicting the perceptual scales. We compare the prediction accuracy on perceptual features using different computational features. Table 2 shows the results, which indicates

Table 2: Comparison of prediction rates (%) of 12 perceptual features.

Features	LBP	Gabor	LBP+Gabor	PCANet-2	PCN-2
contrast	88	91.9	92.7	97.3	**99.8**
repetition	85.7	86.4	89.4	97.5	**99.7**
granularity	82.1	86.0	88.0	97.1	**99.2**
randomness	87.6	88.0	90.8	97.2	**99.2**
roughness	87.1	87.8	91.4	97.5	**99.8**
feature density	87.0	90.3	92.2	97.5	**99.6**
directionality	87.7	90.9	93.1	98.0	**99.8**
complexity	84.9	86.4	90.0	99.3	**99.4**
coarseness	88.1	88.9	90.9	95.9	**99.4**
regularity	85.3	88.2	91.4	97.8	**99.7**
orientation	86.9	90.1	93.0	97.4	**99.8**
uniformity	86.5	89.4	91.2	96.8	**99.2**

that PCN features significantly outperform the others, including commonly used Gabor, LBP and the combination of Gabor and LBP features. These results demonstrate that PCN features are powerful and we are able to estimate perceptual scales from PCN features effectively.

Table 3: Testing errors of feature mapping. Mean squared errors (MSE) and squared correlation coefficients produced by regression models

PTS	Mean Squared Error	Squared correlation coefficient
X	0.0001	0.9988
Y	0.0001	0.9987
Z	0.0069	0.8769

5.3. Results of mapping perceptual features to the perceptual texture space

We calculate the Pearson correlation coefficient ($corr$) between the coordinates ($X, Y$and Z) of PTS and the averaged perceptual scales. The results show that X axis is strongly correlated with the features of contrast ($corr = -0.55$), feature density ($corr = 0.66$), structural complexity ($corr = -0.5$) and coarseness ($corr = 0.62$); the Y axis is strongly correlated with the features of granularity ($corr = -0.52$), roughness ($corr = 0.43$) and structural complexity ($corr = -0.45$); the Z axis is correlated with those of repetition ($corr = 0.79$), randomness ($corr = -0.69$), direction ($corr = 0.67$), regularity ($corr = 0.77$), local orientation ($corr = 0.67$) and uniformity ($corr = 0.60$).

According to the correlation results, regression analysis is performed by using SVM to map the correlated features to the PTS. Table 3 shows the mean squared errors (MSE) and squared correlation coefficient for mapping perceptual scales to the coordinates of PTS. These results indicate that given a set of

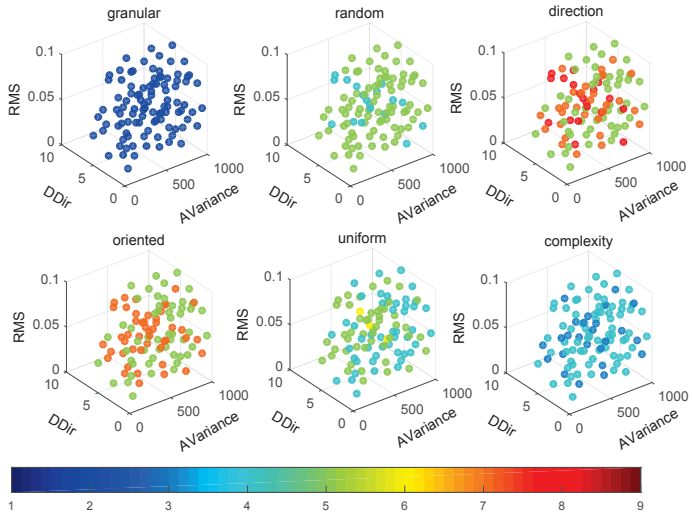

Fig. 5: The plot of the three parameters (RMS, DDir and Avariance) of the Fractal model against six of the twelve perceptual features. Different colors represent different scales specified by the color bar.

out-of-sample perceptual scales, we can accurately predict the intrinsic feature representations in the PTS by using regression models.

5.4. Evaluation of the correlation between the model parameters and the perceptual scales

In order to generate procedural textures with desired perceptual scales, an intuitive way is to investigate the relation between the model parameter values and the perceptual scales. However, for most of the procedural models, multiple parameters need to be determined to achieve the desired texture appearance. Thus, as an example, we evaluate the correlation between the parameters of the Fractal model (Fourier spectral synthesis) and the twelve perceptual scales. Here we choose the Fractal model as the example because it is widely used in computer graphics applications and contains fewer parameters, which make the relation more obvious if there exists any between parameters and perceptual scales. Fig. 5 shows the plot of the three parameters (RMS, DDir and Avariance) of the Fractal model against six of the twelve perceptual features (each set of parameters can generate a texture, and its corresponding perceptual scales are ploted). Different colors represent different scales specified by the color bar.

From Fig. 5, we can observe that the color dots scatter with no explicit patterns in the parametric space of the Fractal model. This suggests that the parameters behave nonlinear with overlapping effects on the output texture appearance. That is to say, the parameter space appears unmanageable and

Table 4: Correlation coefficients (corr) between the three parameters (Avariance, DDir and PMS) of modle Fractal (Fourier spectral synthesis) and the perceptual scales.

Features	Avaiance	DDir	RMS
contrast	-0.2667	-0.0048	0.0128
repetitive	-0.1936	-0.0078	0.13
granular	0	0	0
random	0.1036	0.02	-0.0818
rough	-0.1332	-0.0366	0.0537
density	0.1567	-0.0753	-0.0605
direction	-0.2244	0.0317	0.0642
complexity	0.0727	-0.0154	0.0634
coarse	-0.0010	0.1092	0.0487
regular	-0.1036	-0.02	0.0818
orientated	-0.2126	0.0770	0.0022
uniform	-0.2179	0.0867	0.1029

exploration of the parameter values of a procedural model to achieve a desired perceptual scales would be too complex. Moreover, color variation of the dots in each subplot represents the range of the perceptual scales that possessed by the generated textures when the parameters are traversed. For example, the colors of the dots for the feature "granular" are almost blue. This means that there is almost no change of the feature "granular" perceived by people and the Fractal model may not produce textures with different granular patterns no matter how model parameters change. While for the features "direction" and "oriented", the textures would be perceived differently with various parameter values. We also calculate the correlation coefficients between the three parameters (Avariance, DDir and PMS) of the Fractal model and the twelve perceptual scales. The results are shown in Table 4. The values of the coefficients show there is little relation between the parameter values and the perceptual scales.

In summary, there is no explicit relation between the perceived features and the three parameters of the Fractal model. The proposed framework provides an effective way to generate texture from perceptual descriptions.

5.5. Results of perception-driven procedural texture generation

In this section, we evaluate the performance of the proposed method for procedural texture generation from examples with perceptual description.

First, we report the results on generating a new texture similar to the example but with different perceptual scales. In our experiments, the testing samples are textures generated by the same procedural models used for creating the dataset, but they are not included in the training samples. Various types of textures are tested, including random texture, structural texture and regular texture, representing a large variety of surface appearances. Cosine similarity is

used as a measurement of similarity between the desired perceptual vector and its nearest neighbor in the PTS.

Experimental results are shown in Fig. 6. The input samples are generated by various procedural models and represent different types of textures. The textures in the first column are the inputs textures (IT), and the second column are corresponding models and perceptual scales predicted by using PCN features (PM&PS). Images in the third column (NS1) are the retrieved results by using predicted perceptual scales of the input. The fact that retrieval results are identical to the input implies that the selected perceptual features are effective as texture representations. Then we adjust one of the predicted perceptual scales of the input texture, and the adjusted perceptual scale vector is mapped to PTS using SVM. Images in the fourth column (NS2) are result textures corresponding to modified perceptual scales, i.e. the nearest samples searched in the PTS. In order to evaluate the retrieval results, we compare the adjusted perceptual scales to that of the retrieved textures by computing the cosine similarity. The comparison results are shown in the last column. The horizontal axis represents the 12 perceptual features while the vertical axis represents the scales from 1 to 9. The blue bar represents the adjusted scales and the yellow ones represent the scales of the nearest sample retrieved in the PTS. Samples in the PTS are the training textures annotated with 12 perceptual scales that averaged from the subjects' judgements. They can be assumed as the ground truth. Thus, the value of cosine similarity measures the accuracy of the retrieval by using perceptual scales. The higher cosine similarity means the retrieval results are more in line with human perception.

The results have verified the effectiveness of the proposed approach. As expected, the predicted procedural model is able to generate textures approximating the input image. For example, in Fig. 6, for the input texture A1, the 12 perceptual scales and the corresponding model are predicted. Among them, the perceptual scale of density is 6. Then we change it to 3 while keeping the others unchanged. The fourth column shows the output textures with density equaling to 3. It can be seen that the output results match the perceptual scale we modified while preserving appearance similar to the input image. Moreover, for A1, the cosine similarity is 0.9911, which validate the accuracy and effectiveness of our approach.

In addition, in the experiments, for the input texture, the perceptual scale we choose to adjust depends on the procedural model that generates the input. Liu et.al [2] have verified that textures generated by the same procedural model have relatively similar appearances. Each procedural model has its salient characteristics. That is to say, for one procedural model, the appearances of generated textures only have slight differences in one or a few perceptual features when we vary model parameters. For example, in Fig. 6, the input image C2 is generated by the model of "Fractal (one-over-fBata)", and the perceptually salient characteristics of textures generated by "Fractal (one-over-fBata)" is "directional" and "oriented" [2]. Thus, we choose to adjust the scale of feature "directional". The test results have demonstrated the feature we adjusted is effective.

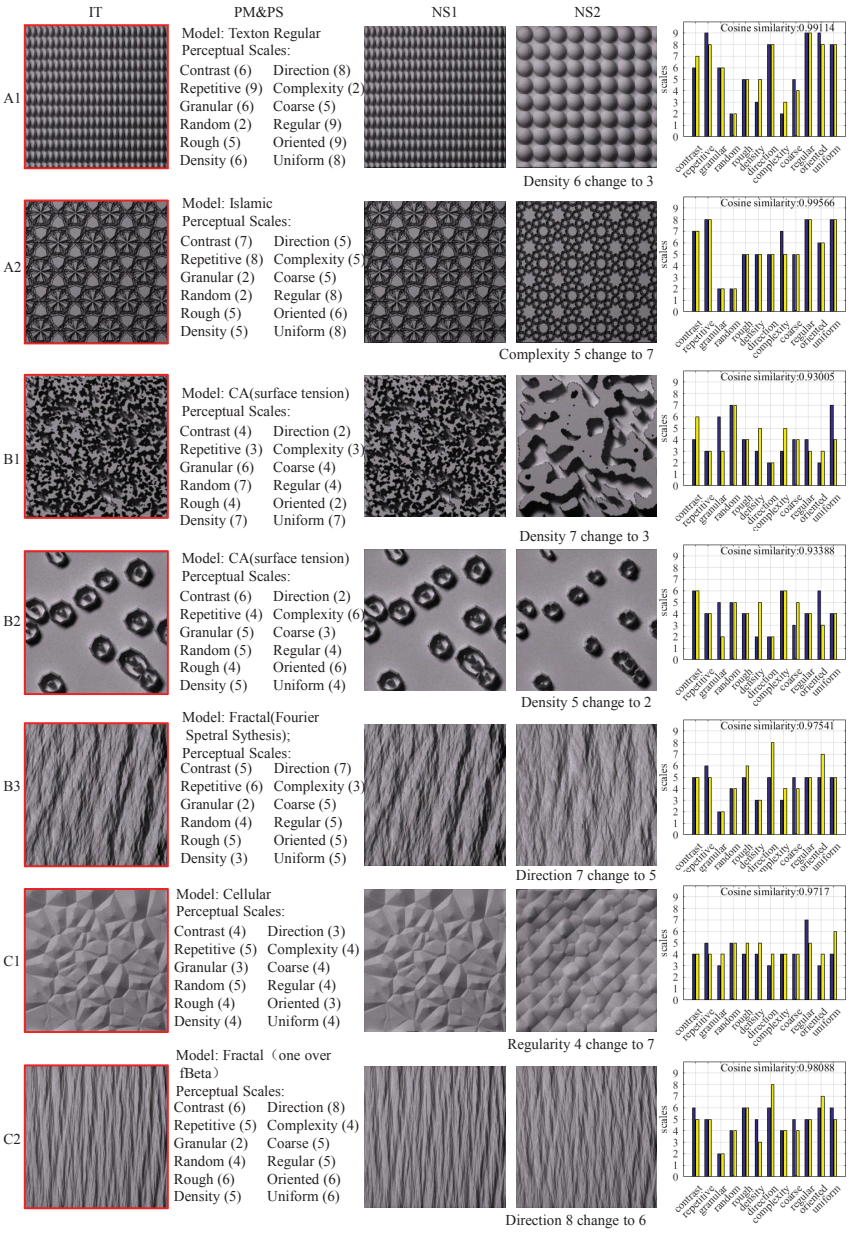

Fig. 6: Generation results of various types of textures. Textures labelled with "A" represent regular textures; textures labelled with "B" represent random textures and textures labelled with "C" represent structural textures. The textures in the first column are the input textures (IT), and the second column are the models and perceptual scales of the input texture predicted by using PCN features (PM&PS). Images in the third column (NS1) are the retrieved results by using predicted perceptual scales of the input. Images in the fourth column (NS2) are the nearest samples searched in the PTS according to the adjusted perceptual scales. The comparison results are shown in the last column. The horizontal axis represents the 12 perceptual features while the vertical axis represents the scales from 1 to 9. The blue bar represents the adjusted scales and the yellow ones represent the scales of the nearest sample retrieved in the PTS.

We then evaluate the performance of the proposed method under different experiment settings. We first predict the perceptual scales of the input texture, and then we adjust the scales of one salient feature, one unobvious feature and two randomly selected features respectively. In addition, we adjust multiple features without the constraint of using the model predicted for the input. Some results are shown in Fig. 7. For example, the input texture "A" is from the model "Folding Texton", and textures in this category consist of randomly distributed near-regular shape elements [2]. When we vary the parameter values of the model "Folding Texton", the generated textures are perceived differently on features "direction", "granular" and "coarse", whereas the scale of feature "random" is perceived almost the same by observers. Texture "A1" is the result when the scale of "direction" is changed from 3 to 8. Obviously, texture "A1" contains stronger directional characteristic compared to the original texture "A". However, when we adjust the scale of "random", the nearest sample "A2" we retrieve in the PTS has no obvious change in the feature of "random". Furthermore, we adjust two features of "granular" and "coarse" together, and the result texture "A3" appears more granular and coarse compared to "A". "A4" is the result texture generated by adjusting two features of "granular" and "coarse" together without the constraint of using the model predicted for the input. The result is still in accordance with the adjusted scales. This suggests that the proposed method is also effective when users adjust multiple perceptual scales simultaneously. However, most of the results produced by adjustment of multiple perceptual scales are difficult to evaluate visually. This is because, unlike adjusting single perceptual attribute, change of multiple attributes may have cross or even more complex effect on texture appearances. Moreover, users should be careful when adjusting multiple features to avoid contradictory situations, e.g. in the case of increasing both the scales of "regular" and "random"; textures perceived with high "regular" and high "random" do not exist.

Finally, we evaluate the performance of the method when natural textures are used as the examples. We select testing samples from PerTex dataset [45]. The PerTex dataset contains 334 textures which are captured from real surfaces. These textures are rendered under constant illumination and viewpoint conditions. In general, the method produces promising results and Fig. 8 shows some examples. The salient features of the retrieved textures are similar to those of the exemplars. However, the appearances of the resulting textures are usually limited by the type of textures that procedural models can produce.

It should be noted that most natural textures are captured with unknown illumination and viewpoint conditions, and illumination conditions can bias the outputs of computational features. In terms of the proposed method, we train the PCA-based convolutional networks by using procedural textures that rendered under the same illumination conditions. Thus, when the exemplar is significantly different from the type of the procedural textures we use, such a fixed networks would not generalize so well. It would be promising to train the PCA-based convolutional networks by using height maps to ensure feature extraction is unbiased by illumination conditions. As a consequence, the height map of the exemplar should be estimated before extracting the PCN features.

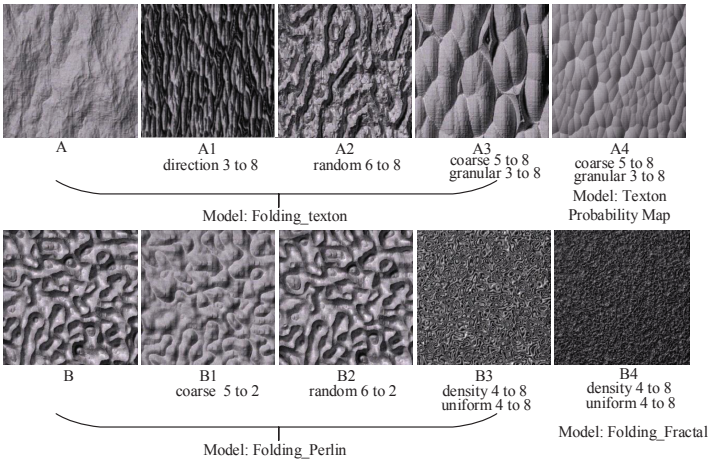

Fig. 7: Results of the proposed method under different experiment settings. The textures in the first column are the input samples, and the textures in the second to the fourth columns are the retrieved results when adjusting the scales of one salient feature, one unobvious feature and two other features respectively. Textures in the last column are the retrieved results when adjusting multiple features without the constraint of using the model predicted for the input.

This might result in more accurate retrieval performance.

In addition, the number of training samples in the PTS affect the accuracy of the retrieval results. In PTS, the distance between two points is a measure of similarity between two vectors of perceptual scales. The closer the points, the more similar the vectors. If the points' distribution is sparse in PTS, the distance between the nearest sample and the given vector of perceptual scales may be not small enough. Thus, the new texture generated with the parameters of the nearest sample does not accord with the given perceptual scales. Theoretically, if the points' distribution is denser, the retrieval result is more accurate. So it is desired to generate more samples by sampling the parameters of the procedural models with a tiny step. However, such a tiny change of the parameters may result in the generated textures perceived to be identical. In addition, more samples will take much more time on rendering process and psychophysical experiments. Therefore, in this study, the training samples are generated by uniform sampling of the whole parameter space for each procedural model, and the samples generated in this way are with obviously different appearances that a human can perceive.

5.6. Evaluation of the performance of PCN

From the experimental results in our task, it can be seen that the proposed PCA-based Convolutional Network is effective for feature extraction. In this subsection, we further evaluate the performance of the PCN in various tasks using publicly available data sets, including hand-written digital recognition, face recognition and texture classification. The two-stage PCN (named PCN-2) is applied to different data sets for simplicity. The final output features of the

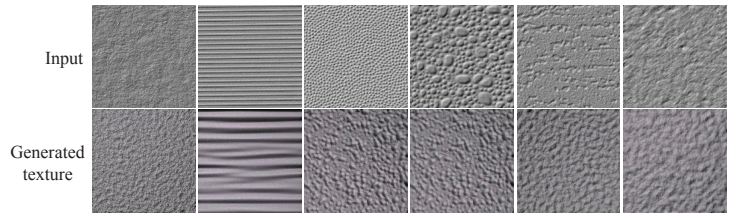

Fig. 8: Retrieval results when the exemplars are natural textures. Top row: the textures from PerTex. Bottom row: the nearest samples retrieved in PTS.

PCN are sent to a linear SVM for classification. We compared the effectiveness and efficiency of the PCN for different recognition tasks.

5.6.1. Hand-written digital recognition on Mnist

We test the performance of PCN on both standard MNIST dataset and basic MNIST dataset[23]. The standard MNIST dataset consists of 60000 training images and 10000 testing images. The basic MNIST dataset is a smaller subset of the standard MNIST, which contains 10000 training images, 2000 validation images and 50000 testing images.

The impact of the number of filters. We first investigate the impact of the number of filters of PCN on the basic MNIST dataset. The patch sampling interval is set as 1 and the patch size is set to 7×7. In the output stage, we set the block size as 7×7, and the block overlapping ratio as 0.5. We select the identity matrix as the indexing matrix, that is, we make every group in the second stage contains only one subset. For one-stage PCN, we vary the number of filters in the first stage from 2 to 16. For two-stage PCN, we first set $L_2 = 8$ and vary L_1 from 2 to 16, and then we set $L_1 = 10$ and vary L_1 from 2 to 16. The results are shown in Fig. 9. We can see that for one-stage PCN (PCN-1), the recognition rates first increase as the number of filters increases and achieves better results when L_1 is set between 7 to 12, then the accuracy decreases with added filters. Obviously, the accuracy of two-stage PCN (PCN-2) is better than PCN-1. For PCN-2, when we set $L_1 = 10$ and vary the number of filters in the second stage, the network has similar performance trend to PCN-1. The dimension of the output features would increase largely with the number of the filters of the last stage, which causes decrease of the accuracy. When we set $L_2 = 8$ and vary the number of filters in the first stage, the accuracy increase for larger L_1.

The impact of the filter size. We examine the impact of the filter size that is also the size of the patch. The parameters of PCN-2 are set to $L_1 = L_2 = 8$, the patch step $k = 1$, the patch size is 7×7, the block size is 7×7, and the block overlapping ratio as 0.5. The identity matrix is selected as the combination way of feature maps. The filter size we considered for each stage is the same. The results are shown in table 5. We can see that the recognition accuracies decrease significantly with larger filter size.

The impact of the block size. We test the impact of the block size for

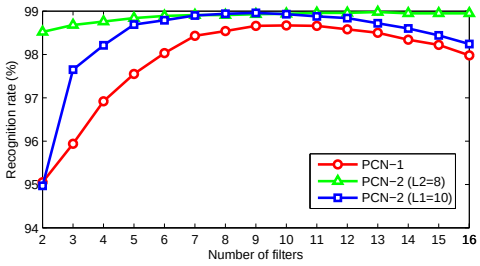

Fig. 9: Recognition accuracy of PCN on basic MNIST for varying number of filters.

Table 5: Recognition accuracy (%) on basic MNIST for varying filter size in PCN-2. The filter size for each stage is the same.

Filter size	3	5	7	9	11	13
Accuracy	97.40	98.82	98.91	98.29	97.07	93.67

histograms in the output stage. The parameters of PCN-2 are set to $L_1 = L_2 = 8$, the patch step $k = 1$, the patch size is 7×7, and the block overlapping ratio is 0.5. The recognition accuracies vary between 98.69% to 98.91% when the block size varies from 4×4 to 13×13, which suggest that the accuracy is less-sensitive to the block size. However, as suggested in [34], larger block size yields robust features against various deformations such as rotation, translation and scaling.

The impact of the block overlap ratio. We also test the impact of the block overlap ratio. The parameters are set the same as the other test. We only vary the block overlap ratio from 0 to 0.9. The recognition accuracies vary between 98.79% to 98.91%. The results also suggest that the performance of the networks is insensitive to the ratio of the block overlap. However, the accuracy achieves the best when the block overlap ratio is set around 0.5.

Comparison with state-of-the-art. We compare the PCN with other state-of-the-art methods both on standard MNIST and basic MNIST. The parameters of PCN-2 are determined by validation. The patch sampling interval is set to 1and the patch size is set to 7×7. In the output stage, we set the block size as 7×7, and the block overlapping ratio as 0.5. We select an identity matrix as the indexing matrix.

We only report the results compared to the methods in which the basic and standard MNIST are used, and do not include the results of methods using augmented training samples with artificially distorted versions of the original dataset. The best result, by using distorted versions of the original training samples, is 99.77% [46].

The results of different algorithms are shown in Table 6. On basic MNIST, we achieve the best accuracy of 99.20% when the numbers of filters in the first and second stage are set to 6 and 11 respectively. On standard MNIST, after validation, the numbers of filters in the first and second stage are set to 8 and 10 respectively, and the accuracy is 99.41%. The results show that PCN-2

Table 6: Comparisons of digit recognition rates(%) of different methods on Basic MNIST and standard MNIST.

Method	Basic MNIST	Standard MNIST
HSC [47]	-	99.23
K-NN-IDM [48]	-	99.46
ConvNet [27]	-	99.47
CAE-2 [49]	97.52	-
ScatNet-2 [28]	98.73	**99.57**
PCANet-2 [34]	98.94	99.34
PCN-2	**99.20**	99.41

Table 7: Face recognition rates (%) of PCN-2 on Extended Yale B dataset for varying the ratio of the training samples (RoS).

ROS	10	20	30	40	50	60	70	80	90
Accuracy	65.37	93.53	96.83	97.44	98.81	99.37	99.45	99.80	100

can achieve comparable performance with the state-of-the-art methods on both datasets.

5.6.2. Face recognition on the extended Yale B dataset

The extended Yale B dataset contains 2414 frontal-face images of 38 individuals[50]. The cropped and normalized 192×168 face images are captured under various lighting conditions.

The impact of the number of the training samples. We next examine the impact of the number of training samples on the Extended Yale B Dataset. We randomly select different number of images as our training samples, and the rest for testing. The number of the training samples varies from 10% to 90% of all the samples. The parameters of the PCN-2 are set to $L_1 = L_2 = 8$, the patch step $k = 1$, the patch size is 5×5, the pooling size is set to 2 for each stage and the block overlapping ratio as 0. The result are tabulated in Table 7. We don't validate the parameters of the network and the images are not pre-processed in this experiment. The performance of the PCN-2 gradually improves as the number of the training samples increases.

We also compare PCN-2 with PCANet-2 with parameter validation. The patch size is set as 5×5, and the numbers of filters in the first and second stage are set as 11 and 8 respectively. The patch sampling interval is set as 1. We use block-overlap ratio is 0.5 and the block size is set to 8×8. The identity matrix is used as the indexing matrix in the second stage. For each subject, we randomly select 30% of the samples as the training samples, and the rest for testing. The experimental results are given in Table 8.

We achieve an average accuracy of 99.7% over 10 experiments, as shown in table 8. Since PCANet has achieved excellent performance on Extended Yale B in the literature, we only make comparison with PCANet in the experiment. The training time of our method including PCN plus SVM is $143s$, and the

Table 8: Face recognition rates (%) and time consumption on Extended Yale B.

Methods	PCANet-2	PCN-2
Accuracy(%)	99.64	**99.70**
Training Time(s)	1753	**143**
Test Time Per Sample(s)	1.29	**0.12**

Table 9: Comparison of texture recognition (%) on DTD.

Methods	SIFT	DeCAF	PCN-2	IFV+SIFT	IFV+DeCAF	IFV+PCN-2
Accuracy	34.70	54.80	38.14	61.2	66.7	58.7

testing time per sample is $0.12s$ based on the same desktop PC. Therefore, it is much more efficient compared to PCANet and has a slightly better recognition accuracy.

5.6.3. Texture recognition on the describable texture dataset (DTD)

We finally evaluate the proposed PCN on the Describable Texture Dataset (DTD) [35]. The DTD contains 5640 images annotated with 47 semantic attributes. Thus, DTD is divided to 47 classes according to 47 semantic attributes with 120 images in each class. Images in DTD are collected from the real-world, especially, it contains textures in the wild. The motivation of the experiment is to explore whether such a simple PCN can work on such a complex database. We did not carry out any pre-processing except for cropping the images in DTD to the same size of 256×256. Because the size of image "studded1_122" is smaller than 256×256, we remove the image from the dataset. The DTD has been split into equally-sized training, validation and test subsets.

The two-stage PCN is used in the experiment. The filter size is set to 7×7; the patch sampling interval is set to 2 and 1 in each stage respectively. The number of filters is set to 16 and 11, and non-overlapping block size is 30×30. The pooling layer is disabled in each extraction stage. The identity matrix is used as indexing matrix in both of the stages. For fairness of the comparison, we conducted two experiments. First, we applied PCN to DTD and used the output of the PCN as features; we then used linear SVM as the classifier. Second, we applied IFV as an encoding to the output of the PCN, and then linear SVM was used as classifier.

The results are shown in Table 9. PCN-2 achieves accuracy 38.14% and is better than SIFT features. When combining with IFV, the performance improves by 20.6%. Although IFV+PCN-2 has a 8% degradation in comparison with IFV+DeCAF, the performance of PCN-2 shown here is still encouraging. Compared to the DeCAF feature, which is based on a complex deep convolutional network trained using ImageNet, the proposed PCN is relatively simple. Once the parameters such as the number of layers, the filter size, and the number of filters are given, only PCA filters need to be learned in an unsupervised way.

6. Conclusion

In this paper, we proposed a novel framework to find procedural texture models and corresponding parameters for generating texture images consistent with user-defined perceptual scales based on an example image. Through extensive psychophysical experiments with a procedural texture dataset, we established a perceptual texture space. The perceptual distance measured in this space improved texture similarity measurement in a manner consistent with human perception. The most similar texture retrieved in the PTS provided parameter values for the predicted texture generation model. Promising results had been obtained and demonstrated the effectiveness of the proposed approach.

Nonetheless, there are some limitations in our current work. The textures in the dataset do not sufficiently cover the full range of parameter space of procedural models we choose. Constrained by the time-consuming rendering process, we only sampled in the parameter space with the purpose of producing visually different textures. In addition, in the psychophysical experiments, the number of samples for the observers is limited due to the available time and labor. The texture samples in the perceptual space are relatively sparse. Moreover, the parameter space is discretized at a roughly uniform scale; it may not reflect all changes in texture perception. The other limitation is that if exemplars are not procedural textures, e.g. a natural photograph, the appearance difference between the photograph and the training samples may dramatically affect the performance of the method. Specifically, the photograph might be taken under unknown lighting conditions, it may affect the accuracy of the prediction of the perceptual scales. The generation result will not be good either.

It should be noted that the proposed framework is general and open to all types of procedural models. When more procedural models are added, the training database will be enlarged and the consistency and stability of the learned PTS will be improved accordingly. The mapping from computational features to perceptual features will also be more accurate. Furthermore, better sampling methods of the parameter values should be investigated in order to optimize the balance between accuracy and efficiency.

Based on the performance of the proposed method, we believe that we are getting close to the actual applications of generating procedural textures according perceptual scales or even semantic description, which is a goal we would like to pursue further.

Acknowledgment

This work was supported by National Natural Science Foundation Of China(NSFC) [grant numbers 61271405, 41576011, 61401413, 61501417, 61601427); Natural Science Foundation of Shandong [grant number ZR2015FQ011]; and Qingdao Agricultural University Research Foundation for Advanced Talents [grant number 1117006].

References

[1] D. S. Ebert, Texturing & modeling: a procedural approach, Morgan Kaufmann, 2003.

[2] J. Liu, J. Dong, X. Cai, L. Qi, M. Chantler, Visual perception of procedural textures: Identifying perceptual dimensions and predicting generation models, PloS one 10 (6) (2015) e0130335.

[3] A. Lagae, S. Lefebvre, R. Cook, T. DeRose, G. Drettakis, D. S. Ebert, J. Lewis, K. Perlin, M. Zwicker, A survey of procedural noise functions, in: Computer Graphics Forum, Vol. 29, Wiley Online Library, 2010, pp. 2579–2600.

[4] S. Padilla, et al., Mathematical models for perceived roughness of three-dimensional surface textures, Ph.D. thesis, Heriot-Watt University (2008).

[5] K. Perlin, An image synthesizer, ACM Siggraph Computer Graphics 19 (3) (1985) 287–296.

[6] J. J. Van Wijk, Spot noise texture synthesis for data visualization, in: ACM SIGGRAPH Computer Graphics, Vol. 25, ACM, 1991, pp. 309–318.

[7] J. P. Lewis, Generalized stochastic subdivision, ACM Transactions on Graphics (TOG) 6 (3) (1987) 167–190.

[8] J.-C. Yoon, I.-K. Lee, Stable and controllable noise, Graphical Models 70 (5) (2008) 105–115.

[9] D. Ghazanfarpour, J.-M. Dischler, Spectral analysis for automatic 3-d texture generation, Computers & Graphics 19 (3) (1995) 413–422.

[10] D. Ghazanfarpour, J.-M. DISCHLER, Generation of 3d texture using multiple 2d models analysis, in: Computer Graphics Forum, Vol. 15, Wiley Online Library, 1996, pp. 311–323.

[11] G. Y. Gardner, Visual simulation of clouds, in: ACM Siggraph Computer Graphics, Vol. 19, ACM, 1985, pp. 297–304.

[12] J.-M. Dischler, D. Ghazanfarpour, A procedural description of geometric textures by spectral and spatial analysis of profiles, in: Computer Graphics Forum, Vol. 16, Wiley Online Library, 1997, pp. C129–C139.

[13] A. Lagae, P. Vangorp, T. Lenaerts, P. Dutré, Procedural isotropic stochastic textures by example, Computers & Graphics 34 (4) (2010) 312–321.

[14] D. J. Heeger, J. R. Bergen, Pyramid-based texture analysis/synthesis, in: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques, ACM, 1995, pp. 229–238.

[15] B. Galerne, Y. Gousseau, J.-M. Morel, Random phase textures: Theory and synthesis, Image Processing, IEEE Transactions on 20 (1) (2011) 257–267.

[16] G. Gilet, J.-M. Dischler, An image-based approach for stochastic volumetric and procedural details, in: Computer Graphics Forum, Vol. 29, Wiley Online Library, 2010, pp. 1411–1419.

[17] L. Gieseke, S. Koch, J.-U. Hahn, M. Fuchs, Interactive parameter retrieval for two-tone procedural textures, in: Computer Graphics Forum, Vol. 33, Wiley Online Library, 2014, pp. 71–79.

[18] M. Jian, K.-M. Lam, J. Dong, L. Shen, Visual-patch-attention-aware saliency detection, IEEE transactions on cybernetics 45 (8) (2015) 1575–1586.

[19] B. S. Manjunath, W.-Y. Ma, Texture features for browsing and retrieval of image data, Pattern Analysis and Machine Intelligence, IEEE Transactions on 18 (8) (1996) 837–842.

[20] T. Ojala, M. Pietikainen, T. Maenpaa, Multiresolution gray-scale and rotation invariant texture classification with local binary patterns, Pattern Analysis and Machine Intelligence, IEEE Transactions on 24 (7) (2002) 971–987.

[21] N. Dalal, B. Triggs, Histograms of oriented gradients for human detection, in: Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on, Vol. 1, IEEE, 2005, pp. 886–893.

[22] D. G. Lowe, Distinctive image features from scale-invariant keypoints, International journal of computer vision 60 (2) (2004) 91–110.

[23] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (11) (1998) 2278–2324.

[24] E. L. Denton, S. Chintala, R. Fergus, et al., Deep generative image models using a laplacian pyramid of adversarial networks, in: Advances in neural information processing systems, 2015, pp. 1486–1494.

[25] G. Berger, R. Memisevic, Incorporating long-range consistency in cnn-based texture generation, arXiv preprint arXiv:1606.01286.

[26] X. Liang, B. Zhuo, P. Li, L. He, Cnn based texture synthesize with semantic segment, arXiv preprint arXiv:1605.04731.

[27] K. Jarrett, K. Kavukcuoglu, M. Ranzato, Y. LeCun, What is the best multi-stage architecture for object recognition?, in: Computer Vision, 2009 IEEE 12th International Conference on, IEEE, 2009, pp. 2146–2153.

[28] J. Bruna, S. Mallat, Invariant scattering convolution networks, Pattern Analysis and Machine Intelligence, IEEE Transactions on 35 (8) (2013) 1872–1886.

[29] M. Henaff, K. Jarrett, K. Kavukcuoglu, Y. LeCun, Unsupervised learning of sparse features for scalable audio classification., in: ISMIR, Vol. 11, 2011, p. 298.

[30] Y. LeCun, K. Kavukcuoglu, C. Farabet, Convolutional networks and applications in vision, in: Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on, IEEE, 2010, pp. 253–256.

[31] T. Serre, L. Wolf, T. Poggio, Object recognition with features inspired by visual cortex, in: Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on, Vol. 2, IEEE, 2005, pp. 994–1000.

[32] J. Mutch, D. G. Lowe, Multiclass object recognition with sparse, localized features, in: Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on, Vol. 1, IEEE, 2006, pp. 11–18.

[33] L. Sifre, S. Mallat, Rotation, scaling and deformation invariant scattering for texture discrimination, in: Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on, IEEE, 2013, pp. 1233–1240.

[34] T.-H. Chan, K. Jia, S. Gao, J. Lu, Z. Zeng, Y. Ma, Pcanet: A simple deep learning baseline for image classification?, arXiv preprint arXiv:1404.3606.

[35] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, A. Vedaldi, Describing textures in the wild, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2014, pp. 3606–3613.

[36] M. Cimpoi, S. Maji, I. Kokkinos, A. Vedaldi, Deep filter banks for texture recognition, description, and segmentation, International Journal of Computer Vision 118 (1) (2016) 65–94.

[37] M. Cimpoi, S. Maji, A. Vedaldi, Deep filter banks for texture recognition and segmentation, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 3828–3836.

[38] T.-Y. Lin, S. Maji, Visualizing and understanding deep texture representations, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015.

[39] A. R. Rao, G. L. Lohse, Towards a texture naming system: identifying relevant dimensions of texture, in: Visualization, 1993. Visualization'93, Proceedings., IEEE Conference on, IEEE, 1993, pp. 220–227.

[40] J. B. Tenenbaum, V. De Silva, J. C. Langford, A global geometric framework for nonlinear dimensionality reduction, Science 290 (5500) (2000) 2319–2323.

[41] J. Liu, J. Dong, L. Qi, M. Chantler, Identifying perceptual features of procedural textures, in: Perception, Vol. 42, Pion LTD 207 Brondesbury Park, London NW2 5JN, England, 2013, pp. 221–221.

[42] A. G. Howard, Some improvements on deep convolutional neural network based image classification, CoRR abs/1312.5402.

[43] M. J. Chantler, The effect of variation in illuminant direction on texture classification, Ph.D. thesis, Heriot-Watt University (1994).

[44] Z. Wang, A. C. Bovik, H. R. Sheikh, E. P. Simoncelli, Image quality assessment: from error visibility to structural similarity, Image Processing, IEEE Transactions on 13 (4) (2004) 600–612.

[45] F. Halley, Perceptually relevant browsing environments for large texture databases, Ph.D. thesis, Heriot-Watt University (2012).

[46] D. Ciregan, U. Meier, J. Schmidhuber, Multi-column deep neural networks for image classification, in: Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on, IEEE, 2012, pp. 3642–3649.

[47] K. Yu, Y. Lin, J. Lafferty, Learning image representations from the pixel level via hierarchical sparse coding, in: Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on, IEEE, 2011, pp. 1713–1720.

[48] D. Keysers, T. Deselaers, C. Gollan, H. Ney, Deformation models for image recognition, Pattern Analysis and Machine Intelligence, IEEE Transactions on 29 (8) (2007) 1422–1435.

[49] S. Rifai, P. Vincent, X. Muller, X. Glorot, Y. Bengio, Contractive auto-encoders: Explicit invariance during feature extraction, in: Proceedings of the 28th International Conference on Machine Learning (ICML-11), 2011, pp. 833–840.

[50] A. S. Georghiades, P. N. Belhumeur, D. J. Kriegman, From few to many: Illumination cone models for face recognition under variable lighting and pose, Pattern Analysis and Machine Intelligence, IEEE Transactions on 23 (6) (2001) 643–660.

Jun Liu received the MSc from Tongji university in 2006, and received the Ph.D. degree from Ocean University of China in 2015. She is currently a lecturer in Qingdao Agricultural University. Her research interests include computer vision, texture perception and machine learning.

Yanhai Gan received the master degree from Ocean University of China in 2014. He is currently an Image Algorithm Engineer with Hisense TransTech Co., Ltd, China. His research interests include neural networks and reinforcement learning.

Junyu Dong received his BSc and MSc from the department of Applied Mathematics at Ocean University of China in 1993 and 1999 respectively, and received his PhD in November 2003 in Heriot-Watt University, UK. He is currently a professor and the head of the Department of Computer Science and Technology in Ocean University of China. His research interests include computer vision, underwater image processing and machine learning, with more than 10 research projects supported by NSFC, MOST and other funding agencies. He has published more than 100 journal and conference papers.

Lin Qi received his BSc and MSc degrees from Ocean University of China in 2005 and 2008 respectively, and received his PhD in computer science from Heriot-Watt University in 2012. He is currently an associate professor in the Department of Computer Science and Technology in Ocean University of China. His research interests include computer vision and visual perception.

Xin Sun received his BSc, MSc and PhD from the College of Computer Science and Technology at Jilin University in 2007, 2010 and 2013 respectively. He did the Post-Doc research (2016-2017) in the department of computer science at the Ludwig Maximilians University of Munich. He is currently an associate professor at Ocean University of China. His current research interest includes machine learning, computer vision and underwater image processing.

Muwei Jian received the PhD degree from the Department of Electronic and Information Engineering, The Hong Kong Polytechnic University, in October 2014. He was a Lecturer with the Department of Computer Science and Technology, Ocean University of China, from 2015 to 2017. Currently, Dr. Jian is a Professor and Ph.D Supervisor at the School of Computer Science and Technology, Shandong University of Finance and Economics. His current research interests include human face recognition, image and video processing, machine learning and computer vision. Prof. Jian was actively involved in professional activities.

Lina Wang is received her MSc degrees from Ocean University of China in 2017. She is currently an engineer in China Unicom Institute of Software. Her research interests include computer vision and machine learning.

Hui Yu received the Ph.D. degree from Brunel University, UK. He is a Reader with the University of Portsmouth, UK. His research interests include vision, computer graphics and application of machine learning to above areas, particularly in human machine interaction, image/video processing and recognition, virtual reality, 3D reconstruction, robotics and geometric processing of human/facial performances. He is Associate Editor of IEEE Transactions on Human-Machine Systems.