EUROGRAPHICS 2001 / A. Chalmers and T.-M. Rhyne (Guest Editors)

Volume 20 (2001), Number 3

Modeling Stochastic Dynamical Systems for Interactive Simulation [†]

L.-M. Reissell and Dinesh K. Pai

Kinesynth Research and University of British Columbia
e-mail: lmr@kinesynth.com

Abstract

We present techniques for constructing approximate stochastic models of complicated dynamical systems for applications in interactive computer graphics. The models are designed to produce realistic interaction at low cost.
We describe two kinds of stochastic models: continuous state (ARX) models and discrete state (Markov chains) models. System identification techniques are used for learning the input-output dynamics automatically, from either measurements of a real system or from an accurate simulation. The synthesis of behavior in this manner is several orders of magnitude faster than physical simulation.
We demonstrate the techniques with two examples: (1) the dynamics of candle flame in the wind, modeled using data from a real candle and (2) the motion of a falling leaf, modeled using data from a complex simulation. We have implemented an interactive Java program which demonstrates real-time interaction with a realistically behaving simulation of a cartoon candle flame. The user makes the flame animation flicker by blowing into a microphone.

1. Introduction

Modeling the physical behavior of natural dynamical systems is in general an extremely difficult task. Consider a typical scenario: animating the behavior of a candle flame in the wind. In current practice, the available alternatives are (1) use motion capture from a real flame and (2) develop a physically-based model which can then be simulated. Motion capture can provide realism but suffers from being difficult to retarget for different conditions or inputs (such as wind speed, direction, etc). This is particularly important for interactive applications such as games, in which the animation has to be generated in real time, and should change based on user input.

Physically-based modeling is an attractive alternative because inputs can be incorporated in the simulation, and it can be easy to produce realistic behavior in some cases; notable successes are in the area of rigid body dynamics [4, 20], cloth [5, 21], fluid flow [26, 12], fire [27, 7] and fracture [17]. However, the physical laws governing the real behavior of everyday objects are complex; in the example of the candle, involving combustion, and turbulent flow. Perhaps the most difficult task is actually choosing the parameters of the model to make the flame behave like a particular candle flame; one typically resorts to tedious parameter tweaking to get the desired behavior.

In this paper, we present an alternative approach based on two key observations:

First, the human perceptual system can not distinguish between large classes of behaviors, and is sensitive to certain statistical properties of the behavior. Therefore exact physical modeling may not be necessary and can be replaced by a statistically equivalent model that produces *plausible* behavior. This is analogous to the visual texture [29, 9, 24], and may be considered a type of *behavior texture*. One benefit of using

[†] This work was supported in part by a research contract from MathEngine PLC.

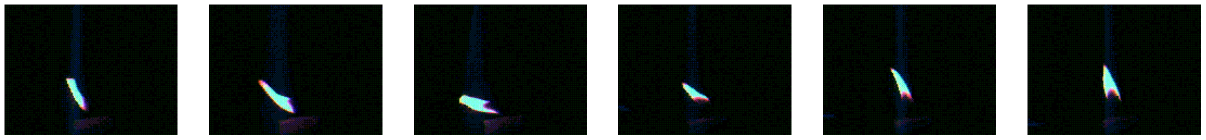

Figure 1: *Film strip of a flame behavior under wind.(The camera exposure is set so that only the flame is visible, for ease of modeling.)*

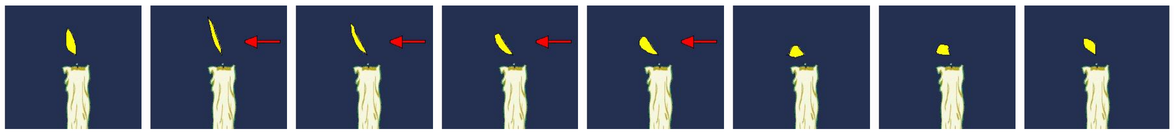

Figure 2: *Film strip of flame behavior under wind simulated using our method.*

statistical models is that they can vastly reduce the time required for simulation.

Second, the statistically correct model can be learned by capturing both the motion and the inputs of a real system. This *reality-based modeling* approach [18] addresses the "parameter tweaking" problem faced by physically-based modeling; it also allows general "black box" models to be developed easily. We can use system identification techniques[15] to approximate the system to the required level of detail (e.g., using linear systems or neural nets [25]). Reality-based modeling techniques constitute a promising new approach in computer graphics, and will provide an alternative to physical modeling in situations where the latter is too complex a task.

Figure 1 shows a video of a real candle flame. Using the techniques described below, the flame outline data was automatically extracted from the video, and a dynamical system model (ARX) of the flame outline as a function of the wind speed was developed. A flame simulated by applying the model to analogous input is depicted in Figure 2. The simulated behavior is very similar to the actual flame behavior. An accompanying video (flamevalidation.avi) which superimposes real candle video with simulated images obtained from exactly the same input data shows that the simulated motion is a good approximation of the real motion. The simulation is very fast compared to simulating the dynamics of the flame: the cost of calculating the parameters which determine the flame position in a given frame of the synthesized video is under 200 flops/frame. Our implementation, in Java, allows the user to interact with the animated cartoon candle flame by blowing on a microphone (see [28]).

Fig. 3 shows another example: the schematic motion of a falling leaf. It is modeled using a discrete stochastic model (Markov chain). The figure shows the original output of a non-linear physical model of the falling leaf, and compares it to synthesized behavior. The motion is qualitatively similar, and the synthesized motion is realistic. However, the cost of the synthesized example is 31 flops/frame, while the cost of simulating the output of the actual dynamical system, which requires the use of a stiff integrator, is a much higher 9239 flops/frame.

1.1. Overview

We construct two kinds of stochastic models: continuous state and discrete state models. Each of these types admits a variety of specific models. In this paper we use ARX models (Autoregressive Models with eXogenous inputs) as our continuous state models. We show how these models are acquired from video and audio data of blowing on a candle flame in Section 3. We use Markov chains as our discrete state models, using them to model the motion of a falling leaf in Section 4. Each state of the Markov chain corresponds to a smooth motion of the leaf on a longer time interval, or a *mode* of behavior, with stochastic transitions between the modes. A hybrid ARX/Markov model is then used to capture the effect of wind gusts on the leaf. Background on these models is provided in Section 2. Section 5 describes our experimental results with the two types of models.

1.2. Related Work

There has been a recent surge of interest in constructing approximate models of complex dynamical systems. These include neural networks and other network architectures [11, 23, 19], image-based and video-based techniques [24, 16, 3]. In control theory, where similar issues arise [15], these methods are called system identification techniques. Stochastic processes have

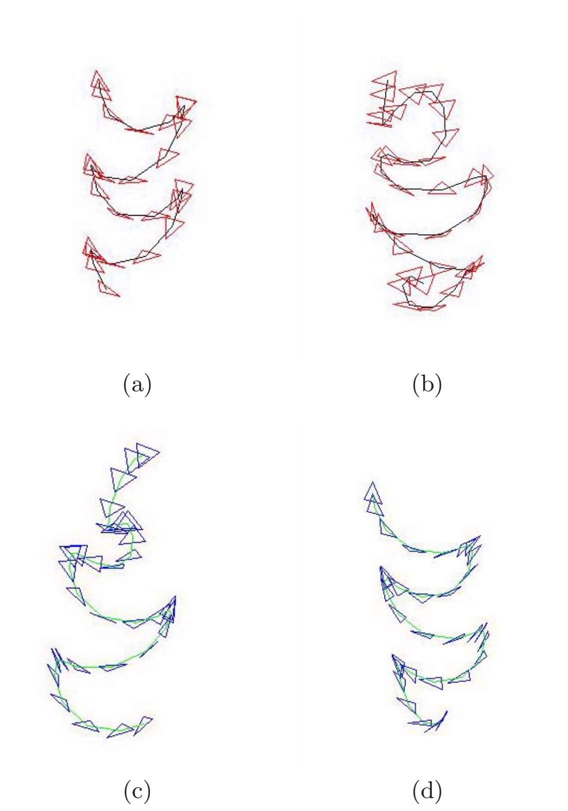

Figure 3: *Original dynamical system output vs synthesized output. (a)-(b): Original motion (c)-(d): Synthesized motion. The motion is qualitatively similar.*

been used in graphics for instance for terrain and texture generation [29, 8, 10].

We now discuss two important recent papers. An excellent example of approximate modeling is the NeuroAnimator [11], which constructs neural network models of physical systems. It shares with our approach the construction of input-output models of a system. Sufficiently large neural networks are universal function approximators, and therefore they could, in theory, approximate complex dynamics well, though the complexity of the network grows rapidly. Unlike our work, the models in NeuroAnimator are deterministic, not stochastic, and therefore complex behavior like a flickering flame is hard to produce (though stochastic models could be constructed with neural networks). In addition, they use only physical simulations and not real world data.

An excellent example at the other extreme is the recent work on Video Textures [24]. As in our work, a stochastic model is constructed using real data in the form of a video; in their case, the model directly uses each frame of the video as a state in a Markov chain, and estimates the transition probablities using similarity at the image level. In contrast we use higher level structures, corresponding to motion segments. Unlike our work, their model does not capture any relationship between input and output; indeed since there are no high-level structures beyond video frames, it difficult to define what an input would be. This makes the video textures good for autonomous simulations, but unsuitable for interaction. Even though specific textures can be made to respond to input (as in the mouse-controlled fish in [24]) this response is hand coded and not learned from the data.

Our approach of using Markov chains to directly model larger segments, or modes, of behavior differs from methods which approximate the underlying dynamical system as a Markov chain where the states correspond to each time instant. The latter are used, for example, in applications of Hidden Markov Models (HMM) to speech and gesture recognition [22], and synthesis [6], as well as in the work in video textures [24]. Nonlinear changes in behavior, such as in the falling leaf example, are difficult to model with a linear model which only considers instantaneous system values.

2. Background: ARX and Markov Chains

We choose relatively simple linear models because they are fast and adequate for many applications. They have robust estimation algorithms, requiring less tweaking than nonlinear methods. Synthesizing behavior will be extremely fast and well suited to real time processing; in ARX models the filtering step used to produce an estimate at each time step requires $O(n)$ time, where n is the length of the filter.

Piece-wise linear models can be used to model nonlinear systems. Such systems can be thought to be in successive "Markov states" which we call *modes* (to avoid confusion with states of the ARX model, etc.). The falling motion of a leaf fits into such a category. This Markov model estimation problem consists of two stages: we first segment system output into segments, then cluster these into modes each representing "similar" behavior, and finally estimate the Markov chain transition probabilities from this classification. Once the model has been learned, new, simulated motion can be synthesized fast by obtaining realizations of the Markov chain. Each mode in this model can correspond to a dynamical system. The techniques, ARX modeling and Markov chains can also be combined to allow for input modeling. Here we use ARX models to determine the effect of wind gusts on the leaf Markov chain motion.

2.1. ARX Models

We briefly review the system modeling techniques used, focusing on the ARX (AutoRegressive with eXogenous input) model. For details, see [15].

A general linear ARX model is expressed with the following equation:

$$A(q)y(t) = B(q)u(t - n_k) + e(t).$$

Here $y(t)$ denotes system output at time t, with dimension n_y; $u(t)$ is the system input, with dimension n_u and time delay n_k; and $e(t)$ is noise, with a specified probability distribution. q is the *forward shift operator* $qu(t) = u(t+1)$, and q^{-1} the corresponding *delay operator*. The matrix $A(q)$ is a $n_y \times n_y$ matrix with entries polynomial in the delay operator q^{-1} and $B(q)$ is a $n_u \times n_u$ matrix. That is:

$$A(q) = I_{ny} + A_1 q^{-1} + \ldots + A_{n_a} q^{-n_a}$$

$$B(q) = B_0 + B_1 q^{-1} + \ldots + B_{n_b} q^{-n_b}$$

The numbers n_a and n_b give the *orders* of the ARX models.

In the multivariate case all coefficients A_i are $n_y \times n_y$ matrices and the B_i are $n_y \times n_u$. In this case, the orders n_a and n_b form $n_y \times n_y$ and $n_y \times n_u$ matrices of integers, respectively. The output component i is related to output component j by a polynomial of order $n_a(i,j)$ in q^{-1}. The delays form an $n_y \times n_u$ matrix n_k. (In practice, when the components of the system behave similarly, the order and delay matrices may often be taken to be constant matrices.)

Written out explicitly, the ARX model is:

$$\begin{aligned} &y(t) + A_1 y(t-1) + \ldots + A_{n_a} y(t - n_a) = \\ &\quad B_1 u(t - n_k) + B_2 u(t - n_k - 1) + \ldots \\ &\quad + B_{n_b} u(t - n_k - n_b + 1) + e(t). \end{aligned}$$

In the multivariate case all coefficients are matrices, as before.

The ARX model represents a filtering process, with a transfer function $G(q) = \sum_{k=1}^{\infty} g(k)\ q^{-1}$. A more general form of such a linear model is expressed in terms of the transfer function of the system, as well as a corresponding function for the error:

$$y(t) = G(q)u(t) + H(q)e(t).$$

Other linear models can be derived from this formulation. In the ARX case,

$$\begin{aligned} G(q) &= \frac{B(q)}{A(q)} \\ H(q) &= \frac{1}{A(q)} \end{aligned}$$

where $A(q)$ and $B(q)$ are the polynomials above, and the orders of the model are determined by the zeros and poles of the transfer function.

2.2. Markov Models

Linear methods can be extended to qualitatively model nonlinear situations, where the behavior changes discretely. Such systems can be thought to evolve in successive *modes*, i.e., "Markov states", where each mode can be modeled by a simple system. We use Markov chains to model such *global* system modes. The use of Markov chains enables us to qualitatively model system behavior.

We use a two-level model for system behavior. On the basic, higher level, the model describes the evolution of system modes. On the lower level, the specific behavior at each time instance in each mode is modeled using a function which represents system output $y(t)$ in the mode (this function depends, e.g., on the initial values in effect). This function can be defined by a differential equation, a linear system, or even samples from the original system outputs. It should be noted that, unlike in traditional uses of Markov chains in graphics, the chain here does not govern instantaneous behavior, but the way the system passes from one mode of behavior to another.

Fig. 4 gives a schematic view of the steps in the representation of leaf motion modes as a Markov chain.

Our model for piecing together segments of system behavior is a finite-state, discrete-time Markov chain $(X_t), t = 1, 2, \ldots$, a stochastic process where the probability $P(X_{t+1} = s)$ depends only on a previous state history $x_t, x_{t-1}, \ldots, x_{t-h+1}$ of fixed length h ($h = 1$, commonly). (For more on stochastic processes and Markov chains, see [14].) The random variable X may take a value from a discrete collection of states $S_i, i = 1, \ldots, K$, which model the modes of the underlying system.

The Markov chain transition probabilities $p_{ij} = P\{X_{t+1} = S_j | X_t = S_i\}$ from state S_i to state S_j are given by the transition matrix $P = (p_{ij})$. We must require that each row of P sums to 1: $\sum_j p_{ij} = 1$.

A realization of the stochastic process (X_t) is obtained by starting with an initial state at time $t = 0$. If the chain is at state S_i at time t, the next state S_j at time $t+1$ is determined randomly according

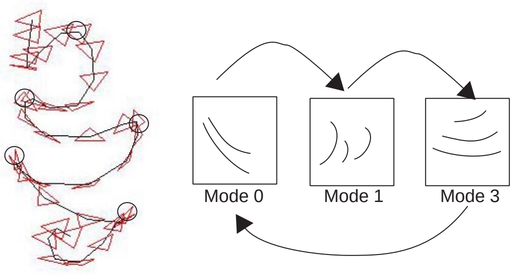

Figure 4: *Obtaining a Markov chain out of falling leaf motion data. Leaf motion is segmented at points of stalling (marked by circles) and the segments are clustered into modes of similar motion, forming the states of the Markov chain.*

to the discrete state probabilities given by the row $P_i = \{p_{ij} : j = 1, \ldots, K\}$ of the transition matrix $P = (p_{ij})$.

In our model, each state S_i in the Markov chain represents a class of segments of the dynamical system output. We define a *realization* of a state S_i as a function $x(a, T)$, dependent on parameters a, on some (not necessarily fixed) time interval T. This function gives a segment of the actual synthesized motion. The model admits a variety of ways of defining the realization x of each state. For example, we can use table look-up of actual data segments, or we can model the states as the output of simple ODEs, as Markov chains, or as splines with a specified knot sequence. In this paper, we will choose this state realization to be simply a randomly chosen example of the original system output segments clustered into state S_i.

Physically plausible synthesized motion can now be generated fast by generating a realization of the Markov chain and a realization of each state.

3. Stochastic Model of Candle Flame

We now describe in detail how a continuous state stochastic model (ARX) can be constructed to simulate blowing on a candle flame. All the steps in processing the raw video and audio into system input and output data are automatic. The system output is obtained using vision methods from video captured with an ordinary desktop camera: the outline of the flame is first obtained from the grabbed frames, then fitted with B-splines to define the output for the ARX model. We capture the input using a microphone to record the wind blown on the candle. The ARX model estimated from this data can then be simulated in real time. We note that the final model estimation procedure is fast (essentially a least squares problem), but the automatic preprocessing of image data can be time-consuming.

3.1. Processing Video Data

The aim is to detect the flame boundary from the intensity (gray scale) image obtained from the original video. We preprocess the image data by converting it directly to a binary image by thresholding. This thresholding is performed from the image intensity histogram by separating the histogram clusters representing the flame and the background and can be automatically computed: a typical candle flame image and its histogram exhibit two sharply defined clusters, as can be seen in Fig. 5.

We then apply an edge detector to the thresholded image to find the flame outline points. Here we use the Sobel detector, which uses the Sobel approximation to the derivative [2]. Since the video has sharp contrasts, edge detection works quite robustly.

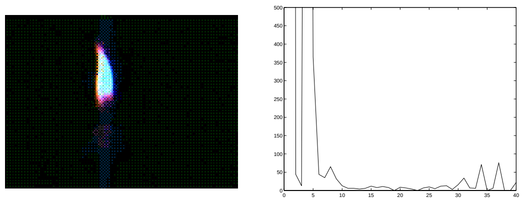

Figure 5: *Flame image and corresponding grayscale image brightness histogram. The flame is represented by the data peaks on the right side of the histogram.*

3.2. Determination of Model Output Data

Raw video or audio data is too noisy and high-dimensional to be used directly used for model parameter estimation. In our candle flame example, the flame image edges are approximated as a B-spline, with a given number N of control points. The dynamical system output for the flame system, at each frame, is then directly taken to be the positions of these N control points. We have used a robust algorithm for the estimation of the control data from acquired images as follows.

The spline estimation algorithm treats the spline as a random curve, defined by control points with a given probability distribution (see [1] for a similar approach).

Our approach processes the cloud of data points sequentially, recursively estimating the posterior probability of the control points. Using a Gaussian approximation to the posterior distribution, this update can be performed very efficiently. The optimal curve estimate is then taken to be the median of the posterior distribution.

3.3. Processing Audio Input Data

We use sound data to estimate the wind input that causes the motion of the flame. The data is recorded at 44.1KHz, and the RMS amplitude of the wind is estimated at video frame rate. The raw amplitude data exhibits background noise, which is removed by finding the mean in the portions of the data with no wind input. Fig. 6 shows the raw amplitude data for a candle movie used in training, as well as the computed frame-by-frame RMS amplitude.

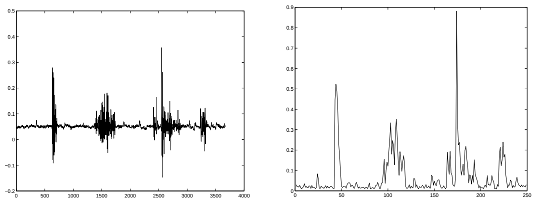

Figure 6: *Sound data. Raw amplitude data at left, frame-by-frame RMS amplitude at right.*

3.4. Model Estimation and Selection

ARX model estimation is done by minimizing the error of predicted data. The regression form of ARX modeling is useful for this purpose. The basic ARX equation (Eq.1) can be expressed as linear regression by using the regression vector - which essentially consists of the system input and output history - $\phi(t) = [-y(t-1), -y(t-2) \ldots u(t-1) \ldots]^T$ and the *system parameters* $\theta = [A_1, A_2 \ldots B_1 \ldots]^T$:

$$y(t) = \theta^T\ \phi(t) + e(t). \tag{1}$$

The regression vector groups together all the past history needed to find the next predicted output value.

The best model is estimated from the data $[y_1, \ldots y_N]$ by minimizing a suitable norm or function describing the error between predicted and observed output values. This *prediction error* is denoted by $V_N(\theta)$ and is typically given by:

$$V_N(\theta) = \frac{1}{N} \sum_{t=1}^{N} |y(t) - \hat{y}(t|\theta)|^2 . \tag{2}$$

For the regression form of ARX modeling this minimization is a least squares problem.

A low order system will usually describe the basic characteristics of the data sets, but often a fairly high order (20 or more) is required for the system to capture the higher frequency dependency on input.

In practice the "best" model is found experimentally by comparing the fits produced by a number of ARX models, either on the original estimation data set, or by *cross validation* on new data sets.

The measures used for evaluating model fit for model M are determined from the differences between actual data and predicted data:

$$J_k(M) = \frac{1}{N} \sum_{t=1}^{N} |y(t) - \hat{y}_k(t|M)|$$

where $\hat{y}_k(t|M)$ is the predicted output of the model M, defined recursively, and, as before, N is the size of the data set $y_1, \ldots, y_N$. If k is finite, we have $k-$step prediction – the value $\hat{y}_k(t|M)$ is determined using all input data and experimental output data from k steps before: $u(t-1), \ldots, u(1), y(t-k), \ldots, \ldots y(1)$. If k is infinite, we have pure simulation from input only.

The *measures for model fit* are now taken to be the expected values of $J_k(M)$, denoted by $\overline{J}_k(M)$.

If the same data is used both for model estimation and for model selection, the error criterion $J_k(M)$ above will be the same as the error criterion (2) used for model identification, and not an unbiased estimate of $\overline{J}_k(M)$. For regression models (1) and $k = 1$ a better estimate can be obtained from estimated data only, by using *Akaike's final prediction-error criterion*:

$$\overline{J}_1(M) \approx \frac{1 + (d_M/N)}{1 - (d_M/N)}\, V_N(\theta).$$

Here d_M is the dimension of the model parameter vector θ. For the derivation of this result, see [15].

3.4.1. Noise model

With ARX models of small order one captures the most significant dynamics of the system, but not fine details. However, in our case using very high order models is not worthwhile because the extra computational expense is usually not justified in terms of the increase in accuracy. Instead we model the residual,

$\rho = y(t) - \hat{y}(t|M)$ as Gaussian noise with variance estimated from the data.

Figure 7 shows the estimated standard deviations for residual error in the position of each control point. Note that there is a significant difference in variance depending on the coordinate direction and also on the location of the control point. Control points in the middle of the figure correspond to the tip of the flame, and have the most variance. We use this estimated variance in our stochastic simulation.

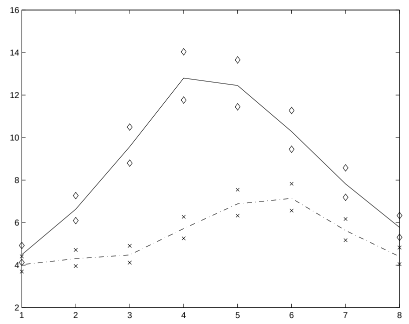

Figure 7: *Plot of standard deviations of residual errors (ordinate) as a function of control point index(abscissa). The top curve (solid) is the standard deviation of the vertical coordinate, while the bottom curve (dashed) is of the horizontal coordinate. The wind blows in the horizontal direction in the training data. The marks above and below the curves show the ends of the 95% confidence intervals.*

3.5. Interactive Simulation of an ARX Model

The core of the simulation algorithm is very simple for an ARX model — essentially a matrix multiplication using the matrix of filter coefficients. Each filtering step in the system takes $n_y \times (n_y \times n_a + n_u \times n_b)$ floating point operations (flops), where n_y is the dimension of the output, n_u is the dimension of the input, n_a and n_b are the orders of the ARX models. Note that the input and output histories should be saved in circular buffers of size n_a and n_b for efficiency.

The computational expense increases linearly with model order; however, it increases quadratically with output dimension n_y, making it computationally very advantageous to decouple model output to the extent possible. Coupled systems can be effectively decoupled by using sparse matrix techniques – coupling is generally strong only for nearest neighbors, and decreases quickly beyond that.

In our flame example we model the behavior of each control point as a separate 2-dimensional system with no visually noticeable loss of accuracy.

4. Stochastic Model of a Falling Leaf

In this section we describe in more detail how to use Markov chains to obtain a qualitative stochastic model for continuous systems which admit discrete modes of behavior, such as the behavior of a falling leaf, and how to synthesize new, realistic animations. The trajectory of a falling leaf is first modeled by a Markov chain whose states are portions of trajectories between leaf stalls (or "modes" of the system). The behavior of the leaf in between stalls is approximately linear and can be captured by a linear system; for simplicity, we choose to model it by using sample trajectories. The results are realistic and computationally fast. See Fig. 3.

The data can be obtained from either measurements of a real leaf or from a physically-based non-linear model of a leaf; here we use the latter, in part to demonstrate that the techniques can be used to approximate complex physical models (as in [11]), and in part because a good physical simulation was available in this case.

4.1. Modeling Using a Markov Chain of Behavior Segments

We develop a stochastic model for qualitative system behavior based on Markov chains in the steps outlined in Fig. 4. Each state in the Markov chain represents a different mode of system output behavior. The states $S_i, i = 1, \ldots, K$ are not known a priori, but are inferred from the data using segmentation and clustering algorithms.

Segmented output is classified using a clustering algorithm to obtain the behavior modes, and we estimate the Markov chain transition probabilities from this classification. Once the model has been learned, new, simulated motion can be synthesized fast by obtaining realizations of the Markov chain.

4.1.1. Segmentation

The output data is first segmented using critical events as demarcations between segments. ("Critical" dynamical system events are used to segment system output in order to capture the dependence of system behavior on such events.) In the falling leaf example, the system modes correspond to the segments of the leaf trajectory between stalls.

The segmentation process results in a collection $\{Y_i : i = 1, \ldots, N\}$ of segments of actual system output, of arbitrary length.

For the falling leaf dynamical system, the important critical values occur as leaf motion stalls (the forward velocity of the origin of the leaf frame of reference in the plane of the leaf becomes zero).

4.1.2. Clustering

After segmentation, unsupervised classification of system output segments $\{Y_i : i = 1, \ldots, N\}$ into states is based on the values of a given (low-dimensional) *feature vector* $X_i, i = 1, \ldots, N$ calculated from output data for each segment Y_i. Feature vectors are then assigned into K classes using the K-means clustering algorithm [13]. These classes constitute the states of the Markov chain and model the modes of system behavior.

We note that the preprocessing for the leaf example is efficient. The slowest step, the K-means clustering algorithm, is in practice generally relatively rapid ($O(d\ n\ k\ i)$, where d is feature vector dimension, i number of iterations, and k number of clusters; however, the number of iterations needed is affected by the initial values chosen.)

4.1.3. Control of Behavior Using the Feature Vector

The choice of components for the feature vector will determine what aspects of the system are captured in the Markov chain behavior, for example overall direction of motion, or motion dependency on end values of previous segment.

Feature vector values should include segment end conditions (at both ends) – in particular, velocity values, since these are crucial in determining system behavior. For the falling leaf dynamical system, we also include feature vector components for segment "size" and overall spatial direction, as determined by the length and the direction vector of the line between segment data endpoints, respectively.

Realistic behavior is obtained here because visually important initial conditions at the beginning of each segment are matched with the appropriate consequences (such as the correct movement direction). In the leaf example, the approximate matching of end conditions at stalls means that it is not necessary to smooth data at joins to produce visually sufficiently continuous behavior.

4.2. Markov Chain Construction

After the clustering of original data segments $\{Y_i : i = 1, \ldots, N\}$, the Markov system states $S_i, i = 1, \ldots, K$ correspond to the classes $\omega_1, \ldots \omega_K$.

The Markov chain $\{X_t\}$ is now obtained simply by finding the state transition probabilities for the transition matrix $P = (p_{ij})$. This is done by counting the state transitions occurring in the data: if M_i is the number of all transitions out of state S_i in the data, and M_{ij} is the number of transitions occurring from state S_i to S_j, we have $p_{ij} = M_{ij}/M_i$.

4.3. Synthesis of New Motion Using Global Markov Chains

Synthesis of new motion is now obtained by first generating a realization of the Markov chain, and then realizing the states obtained with an actual motion segment. The algorithm is outlined below.

- Start with a randomly chosen initial state S_0 at time $t = 0$.
- For each iteration step at time $t + 1$:
  - If the chain is in state S_i at time t, the next state S_j at time $t+1$ is determined randomly according to the discrete state probabilities given by the row $P_i = \{p_{ij} : j = 1, \ldots, K\}$ of the transition matrix $P = (p_{ij})$.
  - The state S_j at time $t + 1$ is realized by random lookup: we choose one of the original system output segments $\{Y_i : i \in \omega_j\}$ clustered into state S_j at random, and denote it by Z_j.
- The final motion is obtained by concatenating the segments Z_j.

We note that smoothing of joins between concatenated segments is not necessary in our falling leaf example, but should be considered in other applications.

4.4. Falling Leaf with Wind Input: Adding an ARX model

Wind gust input is added to the model by using an ARX model to capture the resulting difference in system behavior. We model the change in leaf velocity as a function of wind velocity w and the state x of the leaf:

$$v^w_{CM} - v_{CM} = F(w, x),$$

where v_{CM}, v^w_{CM} denote the velocity of the leaf in windy and windless conditions, respectively. The final effect on leaf position is found by integrating the simulated velocity change. The ARX model is applied locally at each wind gust.

5. Results

5.1. Candle Flame in Wind

The steps needed to construct the dynamical system model from raw video and sound data have been described in Section 3. The processing of video data to extract B-spline shapes was found to be robust and did not require manual intervention. The sound processing was also automatic. In the flame example, we use 8 control points for the modeling, where each control point is governed by its own system (coupled systems were found to not give additional visual benefit).

The best ARX model for each control point was found experimentally by comparing the fits produced by a number of ARX models: a low order system will often describe the basic characteristics of the system, but a higher order is needed to capture some of the higher frequency dependency of data on input. In our examples, the best Akaike FPE model fits are found at fairly high orders (approximately 25). The best ARX model was compared to low order models and the improvement in reproducing the overall behavior of the data was found to be small for visual purposes. In our flame model we have used order 4 for each control point system (number of poles and zeros is 4) with zero delay; the input dimension is 1, and output dimension is 2. Thus, the cost of each simulation step is $8 \times 24 = 192$ flops.

The model reproduces the general behavior of the flame quite well. A video (flamevalidation.avi)[28] demonstrates the results of the ARX model, without the realism-enhancing noise modeling of Sec. 3.4.1. The estimated ARX model is used to simulate flame motion using wind input data from a flame video not used in the estimation of the model. The ARX-simulated flame is shown as a cartoon superimposed on the actual candle video. The simulation captures the essential behavior of the candle flame. In our Java implementation[28], blowing gently or even speaking into the microphone will make the flame move in a realistic way. Blowing harder will extinguish the flame.

5.2. Modeling a Falling Leaf

For controlled experimentation we apply our method to data from a nonlinear differential equation which loosely models the aerodynamics of a falling leaf without wind. The data is obtained using different initial conditions. The motion is represented by the 7-dimensional variable $Y = (q, y)$, where q is the quaternion giving the orientation of the leaf frame relative to world coordinates, and y gives the 3-d position of the leaf frame origin.

The motion mode Markov model is obtained using six segment clusters. The standard Markov history h of length 1 was enough to produce good results, but with some lateral drift; $h = 2$ resulted in qualitatively accurate modeling. Since our method of synthesis is very fast, typically tens of flops per frame, the rapid generation of examples depicting a large number of falling leaves is possible.

Fig. 3(a-d) compares a falling leaf simulated using this dynamical system with an original example. We provide a video containing examples of leaf motion used as training data, and examples of synthesized motion (leafexamples.avi)[28]. Note that the videos do not aim to depict "real" leaf motion but to demonstrate that the synthesized leaf motion captures the behavior of the original, simulated, data. More realistic leaf motion could be achieved by using data from real falling leaves.

Fig. 8 shows the results of adding an ARX model to simulate the effect of wind gusts on leaf behavior, with the actual motion of the leaf depicted in (a), the simulated motion in (b). The motion is shown from top for clarity. The response to wind input is similar in both cases. The ARX model was estimated from different data, and it was not optimized.

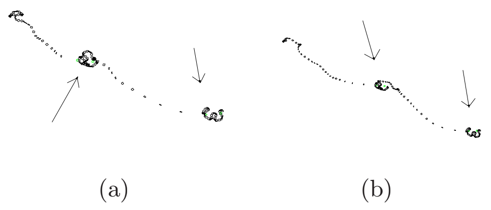

Figure 8: *Leaf motion trajectory in wind shown from top. (Only selected frames are shown.) (a) Original dynamical system output. (b) Simulated motion. Wind gusts (to the left) occur during the frames marked in green, pointed to by arrows.*

6. Conclusions and Future Work

We have described an approach for approximate modeling and simulation of natural phenomena that is robust and easy to implement. We believe it is well suited for interactive computer animation in games, e-commerce and CGI applications. The modeling is based on two types of linear modeling techniques (ARX models, Markov chains), and covers a variety of real situations.

We have explored this approach with two examples: (1) a candle flame flickering realistically in "wind" produced by blowing on a microphone, and (2) the motion of a falling leaf. The data required for modeling is obtained from real experiments (flame) or simulation (leaf). We have implemented software for all steps in this process, including a real-time interactive simulation of a cartoon candle flame in Java.

In future work, more powerful modeling techniques will be considered. We also plan to account for the interaction of the objects with their environment and consider issues of motion control. For example, collision detection will have to be performed, and the behavior of a colliding object modified accordingly. Such modifications could be obtained also by learning procedures from real data; however, in some cases (such as collision of falling leaves) it may be simpler to estimate the effect of the collision on the state of the object, and then restart the original animation process using the new, post-collision state as initial conditions. We also note that the flame example has been modeled as motion in an image plane; as one reviewer has suggested, it is possible to extend the modeling to 3D using multiple views.

References

1. Blake A. and M. Isard. *Active Contours*. Springer, 1998. 5
2. D. Ballard and C. Brown. *Computer Vision*. Prentice-Hall, 1982. 5
3. Z. Bar-Joseph. Statistical learning of multidimensional textures. Masters Thesis, The Hebrew University of Jerusalem. 2
4. David Baraff. Interactive simulation of solid rigid bodies. *IEEE Computer Graphics & Applications*, 15(3):63–75, May 1995. 1
5. David Baraff and Andrew Witkin. Large steps in cloth simulation. In *SIGGRAPH 98 Conference Proceedings*, pages 43–54, 1998. 1
6. M. Brand. Voice puppetry. In *Computer Graphics (Proceedings of SIGGRAPH 99*, pages 21–28, 1999. 3
7. Richard Bukowski and Carlo H. Séquin. Interactive simulation of fire in virtual building environments. *Proceedings of SIGGRAPH 97*, pages 35–44, August 1997. 1
8. D.S. (Ed.) Ebert, F.K. Musgrave, D. Peachey, K. Perlin, and S. Worley. *Texture Modeling: A Procedural Approach*. Academic Press, 1997. 3
9. A. A. Efros and T. Leung. Texture synthesis by non-parametric sampling. In *Proceedings of the International Conference on Computer Vision, Corfu, Greece*, September 1999. 1
10. A. Fournier, D. Fussell, and L. Carpenter. Computer rendering of stochastic models. *Communications of the ACM*, 25(6):271–384, 1982. 3
11. R. Grzeszczuk, D. Terzopoulos, and G. Hinton. Neuroanimator: Fast neural network emulation and control of physics-based models. In *Proceedings of SIGGRAPH 98*, pages 9–20, 1998. 2, 3, 7
12. Weimer H. and Warren J. Subdivision schemes for fluid flow. In *Proceedings of SIGGRAPH 99*, pages 111–120, 1999. 1
13. A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988. 8
14. E. P. C. Kao. *An Introduction to Stochastic Processes*. Duxbury Press, 1997. 4
15. Lennart Ljung. *System Identification*. Prentice Hall, 1999. 2, 4, 6
16. S. Moezzi et al. Reality modeling and visualization from multiple video sequences. *IEEE Computer Graphics and Applications*, 16(6):58–63, 1996. 2
17. James F. O'Brien and Jessica K. Hodgins. Graphical modeling and animation of brittle fracture. *Proceedings of SIGGRAPH 99*, pages 137–146, August 1999. 1
18. Dinesh K. Pai. Robotics in reality-based modeling. In *Robotics Research: the Ninth International Symposium*. Springer, 2000. 2
19. M. van de Panne and E. Fiume. Sensor-actuator networks. In *Proceedings of SIGGRAPH 93*, pages 335–342, 1993. 2
20. MathEngine PLC. Fast dynamics toolkit. http://www.mathengine.com/. 1
21. J. O'Brien R. W. Sumners and J.K. Hodgins. Animating sand, mud, and snow. *Computer Graphics Forum*, 18(1), 1999. 1
22. L.R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proc. of the IEEE*, 7(22):257–286, 1989. 3
23. G. Ridsdale. Connectionist modeling of skill dynamics. *Journal of Visualization and Computer Animation*, 1(2):66–72, 1990. 2
24. A. Schödl, R. Szeliski, D. Salesin, and I. Essa. Video textures. In *Proceedings of SIGGRAPH 00*, pages 489–498, 200. 1, 2, 3
25. J. Sjöberg, Q. Zhang, L. Ljung, A. Benveniste, B. Delyon, P. Glorennec, H. Hjalmarsson, and A. Juditsky. Nonlinear black-box modeling in system idenfication: A unified overview. *Automatica*, 31(12):1691–1724, 1995. 2
26. Jos Stam. Stable fluids. *Proceedings of SIGGRAPH 99*, pages 121–128, August 1999. 1
27. Jos Stam and Eugene Fiume. A multiple-scale stochastic modelling primitive. In *Graphics Interface '91*, pages 24–31, 1991. 1
28. `http://www.kinesynth.com/stochastic.` 2, 9
29. L.-Y. Wei and M. Levoy. Fast texture synthesis using tree-structured vector quantization. In *Proceedings of SIGGRAPH 00*, pages 479–488, 2000. 1, 3