Dynamic Texture on Fixed-Point Architectures

Roberto Costantini, Luciano Sbaiz, and Sabine Süsstrunk

School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne (EPFL)
CH-1015 Lausanne, Switzerland

Abstract. Videos representing smoke, flames, flowing water, moving grass, and so on, show a periodic repetition of some basic pattern. The periodicity is not perfect, since each video frame is actually different from the others, but it is enough to perceive a certain regularity, so that these videos are referred as to *dynamic textures*. Recently, one method based on linear dynamic system theory was proposed to synthesize dynamic textures. Textures are represented as the output of a linear dynamic system and synthesis is reduced to matrix multiplication operations. In this report, we study the problem of implementing this method using fixed-point arithmetic, as required in many portable devices, such as PDAs or mobile phones. This is done by jointly evaluating the effect of model coefficient quantization and of fixed-point precision arithmetic, which are both source of errors with respect to the floating-point implementation. Our analysis shows that the fixed-point scheme permits to obtain visual synthesis quality comparable to the more expensive floating-point implementation, with the advantage of requiring far less buffer memory space and permitting to perform faster synthesis.

1 Introduction

Starting from a sample video, there exist two approaches to synthesize dynamic textures. The first approach is non-parametric and consists in collecting different video clips taken from the same texture [1,2] and "fusing" them together such that time junctions are not noticeable. This approach produces high quality synthetic videos, but consecutive synthetic frames cannot be generated on-the-fly and a large amount of memory is needed to store the entire synthetic video.

The second approach is parametric, i.e., it is based on a model of the dynamic texture. The model parameters are estimated in the analysis step and used during the synthesis. Among such methods, the linear model of Soatto et al. [3] showed to be a valid approach to dynamic texture modelling, both for the good synthesis results that it can achieve, and for the clear mathematical framework used for analyze and synthesize dynamic texture. In this model, the dynamic texture is considered as a dynamical system, where each video frame is represented as a point moving on a trajectory in a given space. The analysis part consists in identifying this trajectory using methods borrowed from the system identification community, and producing an estimation of the model parameters. The synthesis consists in using the model parameters to generate synthetic frames by driving

the system with white noise. The following equation represents the model used for the synthesis:

$$\begin{cases} x[k+1] = Ax[k] + Bv[k] \\ z[k] = Cx[k] + D \end{cases} \quad (1)$$

where $z \in \mathbb{R}^m$ is the vector representing a synthesized color image, $x \in \mathbb{R}^n$ is the system state vector, $A \in \mathbb{R}^{n \times n}$, $B \in \mathbb{R}^{n \times n_v}$, $C \in \mathbb{R}^{m \times n}$, and D are the model parameters, $v \in \mathbb{R}^{n_\nu}$ is a Gaussian random vector $\mathcal{N}(0, I_{n_v \times n_v})$, and $m = N \times M$ is the dimension of a single $N \times M$ color video frame. The index k indicates the frame index and is also referred to as the time index. In the original work of Soatto [3] each image is represented using RGB color encoding for the color channels; in [4] $YCbCr$ color encoding is used, instead, since it achieves a more compact color representation than RGB. In this work, we consider as well the $YCbCr$ color encoding to represent each video frame.

In this report we design a fixed-point implementation of the linear model of Eq.(1), specifically tailoring portable devices, where memory and computational costs are severe constraints.

The model coefficients A, B, C, and D of Eq.(1) are real-value matrices. In the synthesis process they are stored in the temporal memory (Random Access Memory or RAM) where synthesis takes place. In a computer, they are usually represented with high precision, i.e., 64 bits or 32 bits, for double and single precision respectively. The size of the buffer memory necessary to store the coefficients can thus be very large. For instance, the size of the matrix C used to synthesize a color texture of size 160×120 pixels would be $160 \times 120 \times 3 \times 20 \times 64$ bits [1], which corresponds to approximately 8.8 MBytes. This is acceptable for general purpose PCs, even though buffer economy is always favorable, but becomes an obstacle in portable devises, because of more severe memory and CPU power restrictions.

One way to reduce the memory size needed to store the model coefficients is to code them using a lossless coding scheme. A classical procedure is to use quantization followed by entropy coding [5]. In the quantization step, the original real value is quantized, i.e., it is represented using a finite (and generally lower) dimension dictionary. In the entropy coding step, the data are coded using a variable length coding scheme, where shorter codewords are assigned to highly probable outcome values. The entropy coder can compress the data by a factor usually variable from 2 to 5 and has also the property that the code is instantaneous and self-synchronizable. This means that the codeword can be decoded as soon as the last digit has been read and that each codeword ends with a specific terminating symbol, which helps synchronization.

The drawback of using an entropy coder is that decoding operations will take some computational time. It can be done one single time before the synthesis starts to recover the coded coefficients, but in this case the temporal buffer used to store them must have the same size as the original, uncoded data. Decoding can be done on-the-fly, i.e., model coefficients can be decoded only when used.

[1] We consider $n = 20$ as a typical state order size value [4].

However, this solution increases the synthesis cost.

A different way to reduce the data size is to consider quantization without entropy coding. The memory necessary needs to be slightly bigger, but no latency will be introduced by on-the-fly data decoding during synthesis. In this report, we study the quantization step in order to find the minimal amount of bits that can be assigned to the model coefficients to obtain a good synthesis quality. We study this from the more general perspective of translating the floating-point synthesis of Soatto's method in a fixed-point architecture. This, in fact, implies both taking care of the quantization of the model coefficients, and of the finite precision of the operations with respect to a floating-point scheme. The study is done independently from the use of entropy coding, which can always be used if process power allows for it.

Section 2 introduces definitions and concepts of quantization, while Section 3 gives some definition of the fixed-point arithmetic. Section 4 describes the design of the fixed-point synthesis and Section 5 is used to evaluate the fixed-point implementation performance on some test video sequences. Finally, Section 6 concludes the report.

2 Quantization

2.1 Definitions

A continuous time-amplitude signal $x(t)$ is defined for time instants $t \in \mathbb{R}$ and can take any real value in a given subset of $\mathbb{R}$. When dealing with digital signals, however, time and amplitude assume a finite number of possible values. Sampling is used to discretize time, while A/D conversion constraints the signal to discrete (quantized) values.

Given a sampled signal $x \in \mathbb{R}$, the quantized signal y is defined in $\mathcal{I} = \{y_1, y_2, \dots y_L\} \subset \mathbb{R}$ according to the following rule [5]:

$$x = y_k \text{ if } x \in \mathcal{I}_k = \{x : x_k < x \leq x_{k+1}\} \text{ for } k = 1, 2, \dots, L \tag{2}$$

where $L = 2^R$ is the length of the dictionary $\mathcal{I}$ and R is the *bit rate*. The values y_k are called the *reconstruction values* and the x_k are called the *threshold* or *decision* values.

The quantized signal can be represented as a function of the original signal, as depicted in Fig. 1. The abscissa shows the values of the real signal x, while the ordinate indicates the reconstruction values of the quantized signal $y = Q(x)$. The first and last threshold levels correspond to $-\infty$ and $+\infty$ respectively, meaning that $y = y_1$ for $x < x_2$ and $y = y_L$ for $x > x_L$. The difference between the quantized signal and the original one is the quantization error $e_q = x - Q(x)$.

To study the properties of the quantization error, x is considered as the realization of a random variable (r.v.) X, with variance σ_X^2 and probability density function (PDF) $p_X(x)$. The quantized signal is a r.v. $Y = Q(X)$ and the

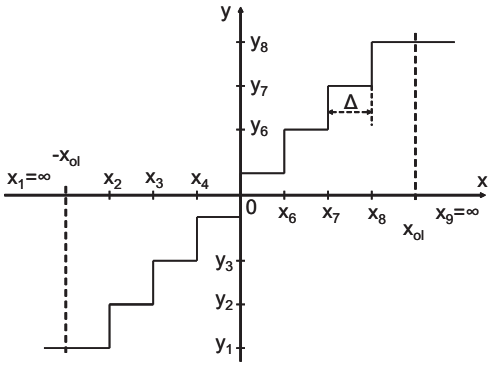

Fig. 1. Quantizer function $y = Q(x)$ in the case of a 3-bit uniform quantizer ($R = 3$).

quantization error is the random variable $Q = X - Y$ of variance

$$\sigma_Q^2 = E[Q^2] = \int_{-\infty}^{+\infty} [x - Q(x)]^2 p_x(x)\, dx = \sum_{k=1}^{L} \int_{x_k}^{x_{k+1}} [x - Q(x)]^2 p_x(x)\, dx \quad (3)$$

This variance defines the quantization noise and is used to compute the signal-to-quantization noise ratio SNR defined as :

$$\text{SNR (dB)} = 10 \log_{10}(\sigma_X^2/\sigma_Q^2). \quad (4)$$

2.2 Granular and overload noise

The quantization noise contains two contributions: *granular* and *overload* noise. Granular noise is the one associated to the quantization of the signal values inside the interval $[-x_{ol}, x_{ol}]$. The term x_{ol}, called *overload amplitude*, is indicated in Fig. 1. It represents the signal value after which the quantized signal assumes the maximum amplitude allowed. When $|x| > |x_{ol}|$ the quantized signal is *saturated*. The overload noise is the one associated to such saturated values, while the granular noise is the noise associated to non saturated values. The first is characterized by burst of values of the same sign, while the second oscillates rapidly between positive and negative values.

If a signal is bounded, the overload distortion can be made zero by imposing $x_{ol} = x_{max}$, where x_{max} is the maximum signal amplitude. For unbounded signals, the overload distortion will always have a contribution, more or less relevant according to the probability that the signal is bigger than the overload amplitude. The ratio between x_{ol} and the signal standard deviation σ_X is a parameter that will be used in the following, and it is defined as the *loading factor*: :

$$f_l = x_{ol}/\sigma_X \quad (5)$$

2.3 Uniform and nonuniform quantization

In uniform quantization, the decision intervals have the same length Δ. When R bits are used, the decision thresholds are defined as $x_1 = -\infty$, $x_{L+1} = \infty$, and $x_{k+1} - x_k = \Delta$ for $k = 2, \ldots, L-1$, where $L = 2^R$.

Once the bit rate R is fixed, the overload value x_{ol} is fixed and defined by Δ and L as $x_{ol} = \Delta L/2$. The granular and overload errors are then defined by x_{ol} as well. If x_{ol} is smaller than the maximum value assumed by the input signal, then we will have some overload noise. On the contrary, a bigger x_{ol} induces a less overload saturation, but a larger granular noise, as Δ increases. The optimum value for x_{ol} depends on the PDF of the signal and can be computed numerically for a given PDF.

In nonuniform quantization, the decision levels have different length. Smaller decision intervals are used when $p_X(x)$ is high, i.e., in the zone where signal values occurrences are more probable. Larger intervals are used for less probable signal values. This introduces a bigger quantization error on a smaller number of coefficients, leading to a globally smaller error variance with respect to a uniform quantizer.

If the signal PDF is log-concave, then a closed formula exists for the optimal quantizer in the minimum-mean-squared-error (mmse) sense [5]. The resulting quantizer is called *Lloyd-Max quantizer* and is defined by the following equations:

$$\begin{aligned} x_k^{opt} &= \frac{1}{2}(y_k^{opt} + y_{k-1}^{opt}) \ \text{ for } k = 2, 3, \ldots, L; \\ x_1^{opt} &= -\infty,\ x_{L+1}^{opt} = +\infty; \\ y_k^{opt} &= \frac{\int_{x_k^{opt}}^{x_{k+1}^{opt}} x p_x(x)\, dx}{\int_{x_k^{opt}}^{x_{k+1}^{opt}} p_x(x)\, dx} \ \text{ for } k = 1, 2, \ldots, L. \end{aligned} \tag{6}$$

The threshold levels are defined as the average value between consecutive reconstruction levels. The reconstruction levels are defined as the conditional expectation of the signal inside an interval given that the signal lays in this interval (centroid): $y_k^{opt} = E(X \mid X \in I_k)$, which is the centroid of the PDF in that interval.

In case of large R, an approximated formula can be found and a compander technique can be use to obtain a nonuniform quantization ([5] p.138).

3 Floating-Point to Fixed-Point Basis

3.1 Definitions

In a given processor, real numbers are represented as *floating-point* numbers, where the scientific notation is used and numbers are expressed as a product between a fractional part and an exponent. The basis for the exponent is 2.

A floating point number can be of *single* or *double* precision, depending if it is represented using 32 or 64 bits, respectively. The IEEE standard floating point

representation [6] divides a floating-point number into three basic components: the sign, the exponent, and the mantissa. The sign (1 bit) indicates the sign of the number, the exponent (8 or 11 bits in single or double precision respectively) the power of two that has to be used, and the mantissa represents the precision bits of the number.

The mantissa is obtained by shifting the point (called *radix point*) of the number in order to push the first "1" number digit in first position before the radix point. For example the number 1101.001110 becomes 1.101001110×2^3, since the radix point has been shifted by three positions toward left. Since the first 1 is always present, it is not considered and the mantissa becomes 101001110.

Differently from floating-point notation, fixed-point numbers are represented using a fixed position for the radix. This is done according to the following bit pattern:

2^{N_I-1}	$\ldots$	2^1	2^0	2^{-1}	$\ldots$	2^{-N_F}
a_{N_I-1}	$\ldots$	a_1	a_0	a_{-1}	$\ldots$	a_{-N_F}

where $N_I - 1$ is the number of bits used for the integer part of the number, N_F is the number of bits used for its fractional part, and the first bit on the left is used for the sign [7]. We call this particular bit pattern $[N_I; N_F]$ format. The number value is equal to:

$$x = -a_{N_I-1}2^{N_I-1} + \sum_{k=-N_F}^{N_I-2} a_k 2^k. \tag{7}$$

If $N_F = 0$, the number is represented with *integer arithmetic*; if $N_I = 1$ the number is represented with *fractional arithmetic*. We can pass from fractional to integer arithmetic by multiplying the number by 2^{N_F}. As an example, the real number 2.23 is represented as $x_q = 2.2299804687500$ in data format $[3; 13]$ (16 bits) and as $x_q = 18268$ using integer arithmetic.

3.2 Fixed-point Arithmetic

In fixed-point arithmetic, operators are defined only among numbers expressed in fixed-point format. In general, fixed-point operations are faster than floating-point ones, since they are performed using less instruction cycles. Tab.1 shows an example of the fixed-point implementation of a simple algorithm. We see that before an operation is performed, the number is quantized to a certain fixed-point format in a block called *quantizer*. This block receives as input a number and a given format $[N_I; N_F]$ and produces as output the quantized value in this format. The quantization is done with a *rounding* operation following the

float a, b, c, y;	a = quant(a,q0);
y = a * b + c;	b = quant(b,q1);
	temp = a*b;
	temp = quant(temp,q2);
	c = quant(c,q3);
	y = temp + c ;
	y = quant(y,q4);

Table 1. Floating point and fixed point implementation of the operation depicted in Fig. 2.

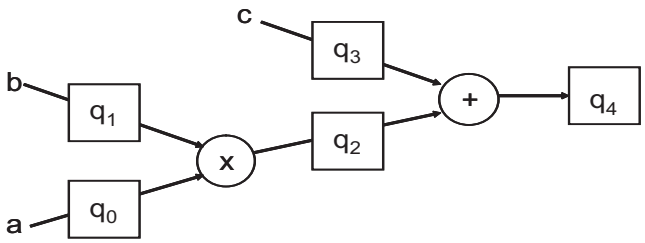

Fig. 2. Example of use of quantizers.

equation:

$$x_q = \frac{\lfloor x \cdot 2^{N_F} + 0.5 \rfloor}{2^{N_F}} \tag{8}$$

Fig. 2 shows a pictorial view of the fixed-point code, where the blocks indicated with the letter "q" correspond to quantizers. We notice that intermediate results between different operation needs also a quantization step, since the operand have to be expressed in the same format. For example, the addition operation imposes the two operands to have the same fractional part. Moreover, after each operation it is generally necessary to discard some information. This happens, for instance, in quantizer q_2 that receives the result of the product of two numbers. In fact, during multiplication between two fixed-point numbers of respectively n_1 and n_2 bits, the results need $n_1 + n_2 - 1$ bits to be exactly represented. In order to avoid a rapid expansion of the number of bits necessary for operations, the result is quantized by discarding the information contained in the least significant bits of the product, thus producing a value with an acceptable number of bits for the next computation.

4 Fixed-point Synthesis Implementation

4.1 Coefficient Quantization

The model coefficients of Eq.(1) are represented using 64 or 32 bit floating-point precision. The first step in implementing the synthesis algorithm using fixed-point arithmetic is to quantize the model coefficients, allowing a lower precision

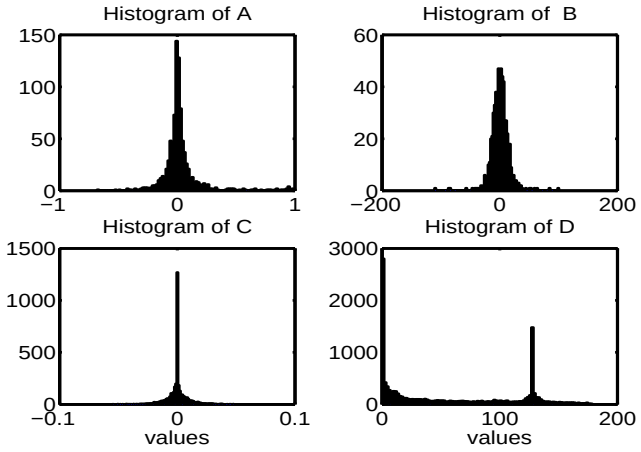

Fig. 3. Histogram of the model coefficients A, B, C, and D obtained from the dynamic texture video "Flame", considering $n = 30$ and YC_bC_r color space.

in order to save memory buffer space and facilitating the computation. In fact, the multiplication of a 32 bit number takes more time than that of a 16 bit number, for instance. In this Section we show how this is done.

As seen in Section 2, quantization depends strongly on the PDF of the signal. In the case of the linear model of Eq.(1), the model coefficients A, B, C, and D have the typical histogram of Fig. 3. The coefficients of matrices A, B, and C have a similar histogram, exhibiting a Laplacian-like PDF, but differ in the dynamic range. The histogram of the vector D is different. This is because this vector collects the temporal average for each pixel value of the video sequence [3]. Since pixels assume values in the range $[0, 255]$, the vector D contains real numbers in this interval. We define $D_q^I = \lfloor D + 0.5 \rfloor$ the quantized D coefficient, where the index I indicates that the vector is constituted by integer values.

More attention is paid to quantize the other matrix coefficients. In fact, we generally do not know in which value range they fall. For this reason, we use scaling to ensure that they fall in an interval that we fix. This is done according to the following formulas:

$$\begin{aligned} A_q &= \frac{\lfloor A \cdot 2^{\kappa_A} + 0.5 \rfloor}{2^{\kappa_A}} = \frac{A_q^I}{2^{\kappa_A}} \\ B_q &= \frac{\lfloor B \cdot 2^{\kappa_B} + 0.5 \rfloor}{2^{\kappa_B}} = \frac{B_q^I}{2^{\kappa_B}} \\ C_q &= \frac{\lfloor C \cdot 2^{\kappa_C} + 0.5 \rfloor}{2^{\kappa_C}} = \frac{C_q^I}{2^{\kappa_C}} \end{aligned} \tag{9}$$

and

$$\begin{aligned}\kappa_A &= R_A - \lceil \log_2(f_l^A \cdot \sigma_A) \rceil + 1 \\ \kappa_B &= R_B - \lceil \log_2(f_l^B \cdot \sigma_B) \rceil + 1 \\ \kappa_C &= R_C - \lceil \log_2(f_l^C \cdot \sigma_C) \rceil + 1 \end{aligned} \tag{10}$$

where κ_A, κ_B, and κ_C are integer values, R_A,R_B, and R_C are bit rates, f_l^A, f_l^B, and f_l^C are loading factors, and σ_A, σ_B, and σ_C are the standard deviations of the model coefficients A, B, and C respectively. The matrices A_q^I, B_q^I, and C_q^I are rounded and scaled model coefficients and are constituted by integer numbers.

In Eq.(9) scaling and quantization are done simultaneously. Scaling is done by multiplying each coefficient by a factor 2^{κ_*}, quantization is done by a rounding operation and a division by 2^{κ_*}. Eq.(10) is used to impose that the integer values lie in a given interval. For example, the scaling coefficient κ_A ensures that the value A_q^I is an integer in the range $[-2^{(R_A-1)}+1, 2^{(R_A-1)}]$. This is why R_A, R_B, and R_C are called rates: they are the number of bits needed to represent the respective integer coefficients. Once the rate R_A,R_B, and R_C are fixed, the problem is to find the values of κ_A, κ_B, and κ_C that ensure the smaller quantization error. This will depend on the loading factor chosen for each coefficient.

As shown in Section 2, the quantization error is composed of two antagonist terms: the granular and the overload error, both depending on the factor load used for quantization. Using a larger load factor has the benefit of decreasing the distortion due to saturation effects, but increases the granular noise. An example of this trade-off is depicted in Fig. 4, where we show the quantization SNR as a function of the loading factor for the matrix B obtained from the analysis of the"Flame" texture.

Fig. 4 shows 3 curves, each obtained considering a given rate (indicated as R_B) for the quantization of matrix B. Note that at $R_B = 16$ we reach the higher SNR using a factor load greater than 4. Using less bits has two consequences: first, it decreases the SNR, since the maximum value is smaller than the maximum value reached by using 16 bit; second, it generally shifts to the left the loading factor value that ensures the minimum error. In fact, for 4 bits, the optimal value for the loading factor is 2. This is because more saturation errors can be tolerated when fewer bits are at disposal, since the most significant part of the signal is centered in zero (see the histograms of Fig. 3): the loading factor has to be smaller to maintain a smaller quantization step Δ when few bits are available. In this first part, the appropriate values for the loading factors are found for each matrix coefficient and the relative scale factors are computed, according to the respective desired bit rates.

4.2 Fixed-point Synthesis Architecture

The second step is to define the correct way of performing the arithmetic operations between the integer-quantized model coefficients. Two constraints have to be considered: the architecture of the fixed-point process that is used and the

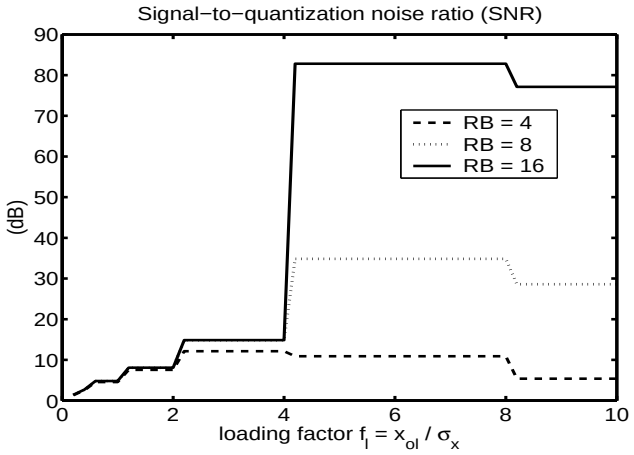

Fig. 4. Quantization error of the matrix B with respect to different values of the loading factor. The matrix B is obtained from the analysis of the video sequence "Flame" ($n = 30$ and $n_v = 20$).

speed of the computation. The architecture fixes the constraint regarding the number and size of the internal registers, while the speed is influenced by the different operations that a processor (or DSP, in general) is capable of performing.

In this work, we do not focus on a particular architecture, but we create a platform that is capable of simulating a fixed-point architecture with different register sizes and general purpose operation between integer data.

The fixed-point implementation of the model of Eq.(1) is the following:

$$\begin{cases} t_1 = A_q^I \cdot X_q^I; \\ t_2 = B_q^I \cdot v_q^I; \\ X_q = (t_1 + t_2) \gg \kappa_A; \\ t_3 = C_q^I \cdot X_q^I; \\ z_q = t_3 \gg \kappa_C + D_q^I; \end{cases} \tag{11}$$

The operations described in Eq.(11) are summarized in the scheme of Fig. 5. Each block denoted as q_* represents a quantization operation of the type:

$$y = Q_I(x) = \lfloor x \cdot 2^{\kappa_*} + 0.5 \rfloor \tag{12}$$

and the operator q_*^{-1} defines the division by 2^{κ_*}, which corresponds to a shifting operation.

The variable X_q indicates the quantized (integer) state variable, the operator "$\gg$" indicated a *shift operator* for integer type data, i.e., a division by a power

of two, and v_q^I is the quantized noise input that drives the dynamical system. The variables t_1, t_2, and t_3 represent registers that store intermediate results.

The first operation is to compute the product $A_q^I \cdot X_q^I$ and store it in a temporary register t_1. Then, the noise is taken into account by computing the product $t_2 = B_q^I \cdot v_q^I$. The two results are then added and scaled by a factor κ_A, since this was the scale factor for matrix A_q^I. Note that we do not separately scale t_1 and t_2 because we chose to impose $\kappa_V = \kappa_A - \kappa_B$. With this choice, only one shifting operation is necessary, since t_1 and t_2 have the same signal level[2].

The noise contribution v_q^I is computed by multiplying the real-valued input noise sequence $v[k]$ of Eq.(1) for 2^{κ_V}, according to the same formula of Eq.(9). Differently for the other coefficients, in this case the rate R_V is imposed by the constraint $\kappa_V = \kappa_A - \kappa_B$, thus having $R_V = \kappa_v - \lceil \log_2(f_l^V \cdot \sigma_V) \rceil + 1$, where the loading factor f_l^V is fixed to ensure that the overload noise is small.

The synthesized image is obtained adding the contribution of t_3 scaled by κ_c to the value D_q^I. The quantization indicated as κ_z of Fig. 5 is defined on 8 bits and corresponds to simple rounding operator that clips the output signal in the interval $[0, 255]$. In fact, the output signal is a digital image whose pixel values are within this range.

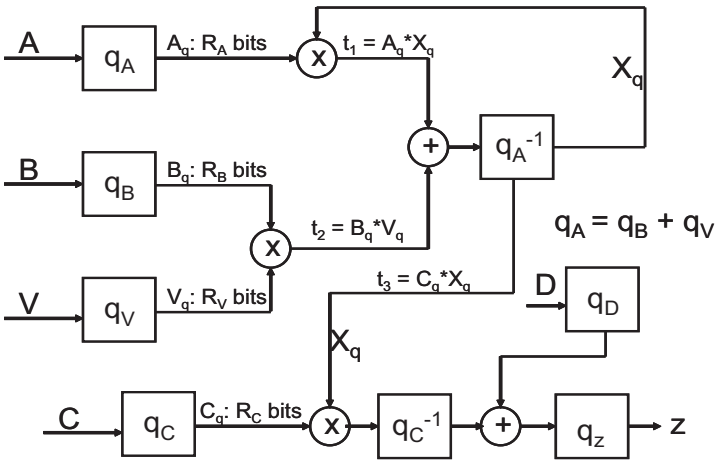

Fig. 5. Schematic representation for the fixed-point implementation of Eq.(1).

[2] In theory, the addition $t_1 + t_2$ could produce an overflow, but in practice this has a small probability. In order to avoid it completely, we can decrease the values κ_A and κ_B by 1.

5 Tests

The scheme of Fig. 5 has been used to test the fixed-point synthesis using different rates for the quantized coefficient A_q, B_q, and C_q. We tested 9 different combinations, obtained by choosing R_A in the set [4, 8, 16], R_C in [2, 4, 8], and fixing $R_B = R_A$. We have chosen lower bit rates for R_C, since we aimed at testing the case where matrix C is quantized using very few bits per coefficient, Cq being the largest matrix to store.

We assigned to the registers t_1, t_2, and t_3 a number of bit equal to $2 \cdot \max(R_A, R_C)$. This permits to avoid overflow during integer operations. The maximum length of the register is reached when $R_A = 16$ and it corresponds to 32 bits, which is the size of a `int` data type in standard ANSI C programming language.

In order to evaluate the synthesis performance, we have considered two criteria. The first is an objective one, based on the computation of the peak-to-signal noise ratio (PSNR) between the synthetic video frames obtained using the fixed-point and the floating-point implementation of the algorithm. In order to have a fair comparison, we used the same input driving noise sequence for both the systems of Eq.(1) and Eq.(11). The PSNR is defined as:

$$PSNR = \frac{1}{S} \sum_{k=1}^{S} 10 \log_{10} \frac{255^2}{MSE(z[k] - z_q[k])} \tag{13}$$

where S is the number of frames considered in the computation of the PSNR. The PSNR evaluates the pixel-wise difference between two images. After a certain time, the synthetic frames $z[k]$ and $z_q[k]$ will differ significantly, since, even though the input noise sequence is the same, the quantization errors present in the fixed-point scheme will act as a disturbance, thus leading to a different input driving noise. For this reason, this is an indicative measure of the synthesis quality for the first few frames that are synthesized, before the quantization error propagates. After this, a visual quality judgement is used, which constitutes the second criterium we used to evaluate the synthesis quality. In fact, from a visual point of view, two synthesized videos can be indistinguishable or comparable, even if their PSNR is very different, since they are not supposed to output the same video. The average PSNR is then computed for the first 10 synthetic frames only, while longer synthetic sequences (100 frames) are created for visual inspection.

During visual inspection we mainly considered two characteristics of the synthetic video created: its dynamic and the visual quality of each frame separately. The visual inspection was performed by the authors of the present report. The web address *http://lcavwww.epfl.ch/∼costanti/fixed-point-results.html* provides the synthetic videos results. The PSNR comparison is given in Fig. 6, where we tested 4 videos sequences, called respectively “Flame”, “Grass”, “Waterfall”, and “Pond”.

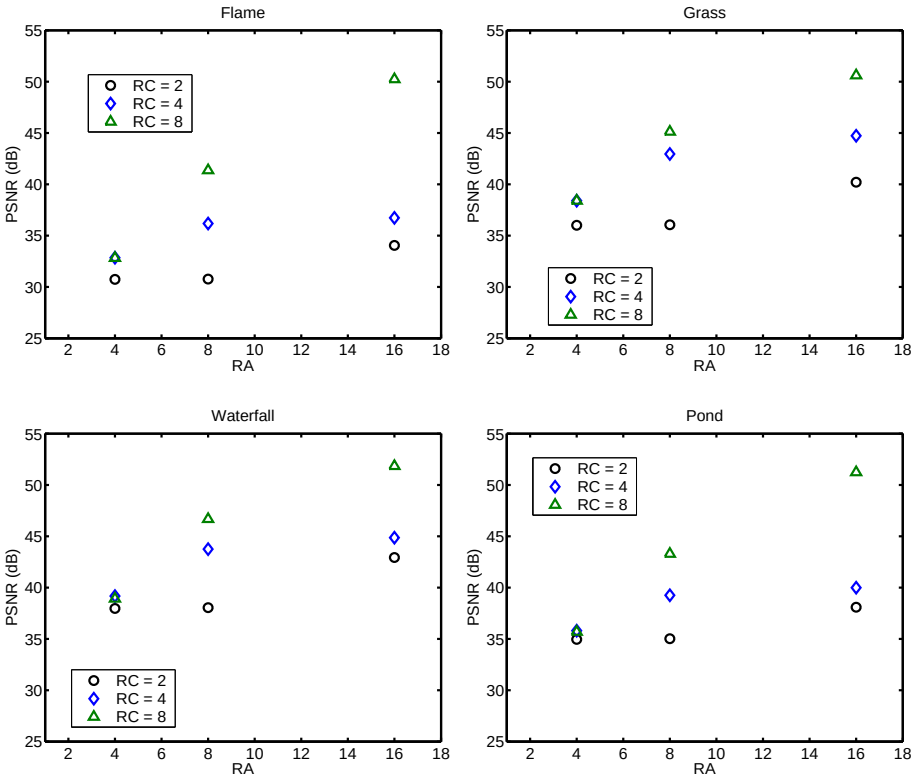

Fig. 6. Performance evaluation using 4 test videos.

For $R_A = 4$, the PSNR value reaches its minimum PSNR, in every test video considered. This corresponds to the worst synthesis result. We notice in this case that the PSNR value does not change significantly when R_C varies from 2 to 8. This means that when few bits are assigned to the quantization of the matrices A and B, the quantized system gives an output whose quality is independent on the quantization of the other parameter C. The visual inspection showed that in this case the synthesis quality was very bad, the dynamic being much affected by the coarse quantization. This is the consequence of the fact that a large error enters in the loop for state vector X_q and thus propagates, affecting the state vector X_q. The role of C_q is limited, since it operates on very noisy data.

For higher values of R_A, the quality of the quantization of C matrix starts to play a role in the overall quality of the synthetic video. We notice, in fact, that for values $R_A \geq 8$, the PSNRs obtained varying R_C are clearly separated. Visual inspection showed that this separation reflects also a perceptually different quality of the synthesized videos. In fact, when $R_A = 16$ and $R_C \geq 8$ the fixed-point synthesis is indistinguishable from the floating-point one, while for

$R_C = 4$ the effect of the quantization are visible on the single image frames, but not on the dynamic.

For $R_A = 8$, the PSNR is in general smaller than that obtained for $R_A = 16$ for some values of R_C and greater for others. In this case, the visual inspection showed that the dynamic of the synthesized video is slightly modified, but still looks acceptable with respect to the dynamic of the original floating-point implementation, and the image quality is good.

Tests have highlighted that the best solution in terms of synthesis quality (both dynamic and visual aspect of the dynamic video) is the one obtained using $R_A = 16$ and $R_C = 8$. From the point of view of coefficient storing, this solution permits to store the model coefficients using 8 times less space than the original floating point coefficients, since just 8 bits instead of 64 are used to store the entries of the matrix C, which has the biggest size among the model coefficients.

If even lower memory is needed, an alternative solution is to consider $R_A = 16$ and $R_C = 4$, which is the second best performance obtained in terms of dynamic and single frame quality. This has been determined by visual inspection, where we noticed that even if the case $R_A = 8$ and $R_C = 8$ has a greater PSNR than the case $R_A = 16$ and $R_C = 4$, the synthesized video obtained using the former quantization is more appealing in terms of visual quality.

Since matrices A and B have much less coefficients than matrix C, our results show that it is always favorable to encode them using 16 bits instead of 8 bits, since the increase in synthesis quality is well justified by a small increase of the number of bits needed to store them.

6 Conclusions

In this report, we studied the problem of implementing the synthesis of dynamic texture in a fixed-point architecture. We have considered a linear model for synthesis, where each synthetic image is represented as the output of a linear system driven by white noise. The fixed-point solution addresses the problem of optimal model coefficient quantization in the sense of memory requirement needed to store them and final synthesis quality. We found that a solution that permits to obtain a synthesis quality indistinguishable from the floating-point solution is the one that assigns 16 bit to the matrices of the autoregressive part (state vector update) of the system and 8 bit to the observation matrix C. This permits to store the model coefficient using 8 time less space than the original coefficients, represented using 64 bit precision.

Suboptimal solution also exists, using only 4 bits for C, for instance. In this case, the dynamic of the video slightly changes, but the overall synthetic video quality is still acceptable.

7 Acknowledgments

This project is supported by the Swiss National Science Foundation (SNF) under grant number 21-067012.01.

References

1. Schödl, A., Szeliski, R., Salesin, D., Essa, I.: Video textures. (Proc. of SIGGRAPH 2000) 489–498
2. Kwatra, V., Schödl, A., Essa, I., Turk, G., Bobick, A.: Graphcut textures: Image and video synthesis using graph cuts. (Proc. of SIGGRAPH 2003) 277–286
3. Doretto, G., Chiuso, A., Wu, Y., Soatto, S.: Dynamic textures. Int. Journal of Computer Vision **51** (2003) 91–109
4. Costantini, R., Sbaiz, L., Süsstrunk, S.: Dynamic texture synthesis: Compact models based on luminance-chrominance color representation. (To appear in Proc. ICIP 2006, Atlanta, GA, USA)
5. Jayant, N., Noll, P. In: Digital Codign of Waveforms. Prentice-Hall Signal Processing Series (1984)
6. IEEE: Standard for binary floating-point arithmetic. (IEEE Std 754-1985)
7. Zamuner, G.: Implementazione in virgola fissa di un algoritmo per la codifica vocale a banda larga su dsp tms320c62xx. Tesi di Laurea in Elaborazione Elettronica di Segnali e Immagini (2002)