Journal of Universal Computer Science, vol. 16, no. 5 (2010), 586-603
submitted: 27/6/09, accepted: 12/2/10, appeared: 1/3/10 © *J.UCS*

On Succinct Representations of Textured Surfaces by Weighted Finite Automata

Jürgen Albert
(University of Würzburg, Germany
albert@informatik.uni-wuerzburg.de)

German Tischler
(Newton Fellow, King's College London, United Kingdom
german.tischler@kcl.ac.uk)

Abstract: Generalized finite automata with weights for states and transitions have been successfully applied to image generation for more than a decade now. Bilevel images (black and white), grayscale- or color-images and even video sequences can be effectively coded as weighted finite automata. Since each state represents a subimage within those automata the weighted transitions can exploit self-similarities for image compression. These "fractal" approaches yield remarkable results in comparison to the well-known standard JPEG- or MPEG-encodings and frequently provide advantages for images with strong contrasts. Here we will study the combination of these highly effective compression techniques with a generalization of weighted finite automata to higher dimensions, which establish d-dimensional relations between result-sets of ordinary weighted automata. For the applications we will restrict ourselves to three-dimensional Bezier spline-patches and to grayscale images as textures.
Key Words: Weighted Finite Automata, image compression, self-similarity, polynomials, Bezier splines, Parametric Weighted Finite Automata, bicubic Bezier patches, textured surfaces
Category: F.1.1, I.3.3, I.3.5, I.3.7

1 Introduction

Many variations of finite state concepts have been studied in depth in computer science and there are famous classical applications, for example the specifications of tokens in compilers for high-level programming languages as in the scanner generators *lex* or *flex* [Aho et al. 2007] or efficient representations of patterns for online-search algorithms as in the KMP-algorithm [Knuth et al. 1977]. In most textbooks on formal languages and automata theory finite automata are treated as acceptors; for any given input sequences they decide whether or not they are members of some specified set. These membership problems for strings are in fact reachability problems for the set of final states from the start state (see e.g. [Wood 1987]).

Thus, finite acceptors can also be viewed as Boolean functions over input sequences. Finite automata are nondeterministic in general, so the Boolean result for an input sequence is *true*, iff there exists at least one accepting transition sequence for it.

Now it is easy to associate bilevel images to input sequences, say *white* if it is *true* that this sequence is accepted and *black* otherwise. If we assume for now that all our bilevel images consist of squares of $2^k \times 2^k$ pixels and the input alphabet $\{0, 1, 2, 3\}$ provides the labels for each quadrant and recursively for each subquadrant then we can address each pixel by a word of length k. This means that the acceptance pattern of a finite automaton for all input sequences of length k yields a bilevel image of resolution $2^k \times 2^k$.

Even extremely simple finite automata can generate fractal (or self-similar) patterns under this interpretation. The best known example is probably the Sierpinski-triangle, consisting of only one state, which is also final, and transitions of this state into itself for the input symbols 0, 1, and 2 but not for the symbol 3.

2 Introduction to Weighted Finite Automata

In the classical papers on automata theory one can find early definitions of automata with transitions labeled by real numbers. The main topic of interest was then to study the probabilistic behaviour of acceptors or to build an even more general framework as in formal power series (e.g. [Eilenberg 1974], [Salomaa and Soittola 1978]). The first papers that sowed the seeds for image compression as an application area for weighted automata are probably [Berstel and Nait Abdullah 1989] and [Culik and Kari 1993]. For a recent collection of research results on weighted automata including weighted finite automata, weighted tree automata and others, see [Droste et al. 2009].

2.1 Definitions

If we consider grayscale images of a resolution of $2^k \times 2^k$ pixels for some arbitrary $k \in \mathbb{N}$ instead of the bilevel cases, it is straight-forward that we can always find a finite automaton with states and transitions weighted by real values which will generate exactly this given image as a real-valued matrix. Just imagine a complete quadtree of depth k with all states and transitions weighted by 1.0 except for the transitions to the leaves of this tree, where each transition carries the weight of the required grayness-level. In the following definitions we will not restrict ourselves to any special image format, instead all kinds of color-spaces (bilevel, grayscale, RGB = red-green-blue, ...) are allowed.

2.1.1 Weighted Finite Automata

In a general setup weighted finite automata (WFA) are finite automata computing functions over semirings, introduced in [Schützenberger 1961]. We will use here the semiring definition as given in [Berstel and Reutenauer 1988] and the symbol $\mathcal{S}$ will denote an arbitrary but fixed semiring endowed with some topology. As every semiring can at least be equipped with the discrete topology, where we have only "isolated points" as elements, this is not a limitation in comparison to other WFA definitions. Additionally, we will often assume an $\mathcal{S}$ over some metric space and we will call such a semiring a metric semiring.

Definition 1. A Weighted Finite Automaton (WFA for short) Z is a quintuple $Z = (Q, \Sigma, W, I, F)$, where

1. $Q = \{0, 1, \ldots, n-1\}, n \in \mathbb{N}^+$ is a finite non-empty set of states,
2. $\Sigma = \{0, \ldots, l-1\}, l \in \mathbb{N}^+$ is a finite non-empty alphabet,
3. $W = (W_0, \ldots, W_{l-1})$, where $W_i \in \mathcal{S}^{n \times n}$ is the matrix of weighted transitions for each input symbol $i \in \Sigma$,
4. $I \in \mathcal{S}^{1 \times n}$ is the initial distribution and
5. $F \in \mathcal{S}^{n \times 1}$ is the final distribution.

The function $f : \Sigma^* \mapsto \mathcal{S}$ computed by Z is given for each $w = a_1 \ldots a_k \in \Sigma^*$ by

$$f(w) = IW_{a_1}W_{a_2}\ldots W_{a_k}F = I\prod_{i=1}^{k} W_{a_i}F \tag{1}$$

which can also be written in detailed form as

$$f(w) = \underbrace{\sum_{p=(q_0,q_1,\ldots,q_k)\in Q^{k+1}}}_{\text{All possible paths } p \text{ of length } k} \underbrace{\left(I(q_0)\prod_{i=1}^{k} W_{a_i}(q_{i-1},q_i)F(q_k)\right)}_{\text{Value computed for path } p} \tag{2}$$

The function $f^\omega(w) : \Sigma^\omega \mapsto \mathcal{S}$ computed by Z is then given for each $w = a_1a_2\ldots \in \Sigma^\omega$ by

$$f^\omega(w) = \lim_{k\to\infty} f(a_1a_2\ldots a_k) \tag{3}$$

We will use notations like Q_Z, Σ_Z, etc. in the following for the set of states of the automaton Z, the input alphabet of the automaton Z, etc.

Let f be the function computed by the WFA $Z = (Q, \Sigma, W, I, F)$, where $Q = \{0, \ldots, n-1\}$, and let further f_i for $i = 0, \ldots, n-1$ be the function computed by the WFA $Z_i = (Q, \Sigma, W, I_i, F)$, which equals Z except for the initial distribution I_i, which is set to the i'th unit vector. We call f_i the state function of state i in Z. Equation 1 yields

$$f_i(av) = \sum_{j=1}^{n} W_a(i,j)f_j(v) \tag{4}$$

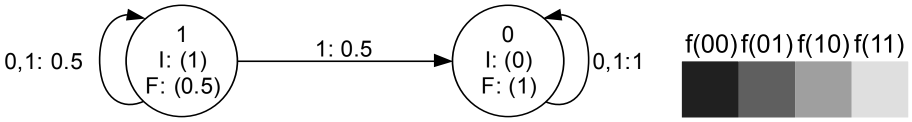

Figure 1: WFA computing $f : \Sigma^* \mapsto \mathbb{R}$, $f(w) = \text{bin}(w) + 2^{-(|w|+1)}$ (left) and values given as grayscale levels for all words w of length 2 (right)

for each $a \in \Sigma, v \in \Sigma^*$. Thus the state function of state i is determined by $|\Sigma|$ linear combinations of state functions. The function computed by Z is a linear combination of the functions of its states.

Now we will give a short survey of results, which are needed for the main chapters of this paper. It was shown in [Berstel and Reutenauer 1988] that a function $f : \Sigma^* \mapsto \mathcal{S}$ is computable by a WFA, if and only if it can be represented as a rational formal power series. Clearly, each function $f : \Sigma^* \mapsto \mathcal{S}$ with finite support is WFA computable and for each $s \in \mathcal{S}$ the functions sf given by $(sf)(w) = sf(w)$ for each $w \in \Sigma^*$ and fs given by $(fs)(w) = f(w)s$ for each $w \in \Sigma^*$ are WFA computable. It is well known (cf. [Berstel and Reutenauer 1988]) that if f and g are WFA computable functions, then so are the functions $f+g$ and $f \cdot g$ defined by $(f+g)(w) = f(w)+g(w)$ and $(f \cdot g)(w) = \sum_{uv=w} f(u)g(v)$ for each $w \in \Sigma^*$. If $\mathcal{S}$ is a commutative semiring, then the function $fg(w)$ given by $(fg)(w) = f(w)g(w)$ for each $w \in \Sigma^*$ is WFA computable (cf. [Berstel and Reutenauer 1988, Culik and Karhumäki 1994]). There exist effective state minimization algorithms for WFA over fields (cf. [Berstel and Reutenauer 1988, Culik and Kari 1997]).

WFA over $\mathbb{R}$ can compute some important classes of real functions over the unit interval, where each input word $w = a_1 a_2 \ldots a_k \in \Sigma^*$ over the binary alphabet $\Sigma = \{0, 1\}$ is interpreted numerically as

$$\text{bin}(w) = \sum_{i=1}^{k} a_i 2^{-i} \tag{5}$$

and for each $w = a_1 a_2 \ldots a_i \ldots \in \Sigma^\omega$

$$\text{bin}^\omega(w) = \lim_{i \to \infty} \text{bin}(a_1 a_2 \ldots a_i) \tag{6}$$

A simple example of a WFA computing a real-valued function is shown in Figure 1. The “one-dimensional” image computed by the automaton using all words of length 2 is also shown in Figure 1. The automaton basically computes a linear function. The function however is not purely linear for words of finite length. This deviation from strict linearity however is on purpose. In the case of finite words we can assign the half-open interval $I(w) = [\text{bin}(w), \text{bin}(w) + 2^{-|w|})$ to each word $w \in \Sigma^*$. The automaton computes the average value of the linear

00	01	10	11
02	03	12	13
20	21	30	31
22	23	32	33

Figure 2: Addressing subimages by words of length 2 over $\{0, 1, 2, 3\}$ in Morton-order

function in the interval $I(w)$ for each word $w \in \Sigma^*$ instead of the value at the left interval border.

The real-valued polynomials play a special role for WFA over $\mathbb{R}$. They are the only smooth functions (that is, every derivative is continuous) computable by WFA to arbitrary precision. Furthermore, WFA can encode these polynomials very efficiently. Just $n + 1$ states are needed for a polynomial of degree n over the unit interval [Culik and Karhumäki 1994, Droste et al. 2006, Derencourt et al. 1992]. As shown below, also polynomials in more than one variable, polynomials defined on the unit-square, the unit-cube. etc., have very succinct WFA-representations.

But WFA over $\mathbb{R}$ can also represent other important classes of non-smooth functions. One of these classes are the dyadic finite impulse response scaling functions and wavelets [Culik and Dube 1997, Tischler et al. 2005].

WFA over $\mathbb{R}$ can be used to compute real functions of dimensions $d > 1$ on $[0, 1]^d$, the unit-cube of dimension d. One prominent example are WFA encoding grayscale or color still-images. There are especially two main concepts. The first is to increase the number of alphabet symbols from 2 in the one dimensional case to 2^d for a space of dimension d. In early papers concerning image compression using real WFA (cf. [Culik and Kari 1993, Culik and Kari 1994, Culik and Kari 1995]) it was common to use an alphabet of four symbols for addressing pixels in images. Thus the interpretation of each word $w = a_1 \dots a_k \in \{0, 1, 2, 3\}$ is given by

$$x(w) = \sum_{i=1}^{k} (a_i \bmod 2) 2^{-i} \tag{7}$$

and

$$y(w) = \sum_{i=1}^{k} \left(\left\lfloor \frac{a_i}{2} \right\rfloor \right) 2^{-i} \quad . \tag{8}$$

This interpretation conforms to the so-called Morton- or Z-order shown in Figure 2.

The second concept consists of keeping the binary alphabet $\Sigma = \{0, 1\}$, but interleaving the components of the coordinates in a single word. Thus we could

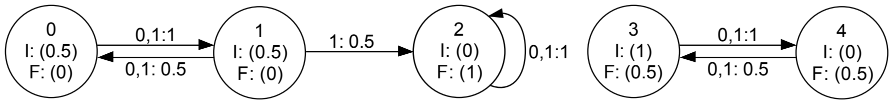

Figure 3: WFA computing $f(x,y) = \frac{x+y}{2}$ and its grayscale image over the unit interval

for instance interpret the word $w = y_1x_1y_2x_2 \dots y_kx_k \in (\Sigma^2)^*$ with

$$y(w) = \sum_{i=1}^{k} y_i 2^{-i}, \quad x(w) = \sum_{i=1}^{k} x_i 2^{-i} \ . \tag{9}$$

Similar to bin^ω we can define y^ω and x^ω for words of infinite length. An automaton computing the function $f(w) = (\text{bin}(x_1 \dots x_k) + \text{bin}(y_1 \dots y_k))2^{-1} + 2^{-(k+1)}$ for each $w = y_1x_1 \dots y_kx_k \in (\Sigma^2)^*$ according to this scheme is shown in Figure 3.

Clearly, for every $k \in \mathbb{N}$ and for every pixel image I of resolution $2^k \times 2^k$ there is now a WFA Z such that evaluating Z for all words of length k, exactly reproduces the image I. As the real numbers form a field and there exist state minimization algorithms for WFA over fields, one can always compute a state minimal WFA representing the image.

Although the state minimal representation of an image looks very attractive in theory, there are two important facts to be considered in applications. The first is that a state minimal WFA frequently is not a minimal WFA-description for its function. Most of the space for the description is usually required for the labeled and weighted edges (cf. [Culik and Kari 1994]). The second is that exact represention of source images is normally not desirable, since most of the time the source contains some kind of noise anyway. Thus, WFA for "lossy" image compression makes more sense for real-world applications. Lossy WFA compression of images, i.e. representing an image approximately by a WFA and compressing the resulting WFA by means of entropy coding, is quite competitive. The results given in [Culik and Kari 1993, Hafner 1999] suggest that WFA compression can outperform the ubiquitous (baseline) JPEG in most applications and in some can even compete with JPEG2000 (cf. [Wallace 1991], [JPEG2000 Core]).

Each state in a WFA represents a subimage of the complete image. Thus each state function can be identified with a state image. Equation 4 tells us that each state image is defined as a set of linear combinations of states in the automaton.

2.1.2 Parametric Weighted Finite Automata

Though WFA represent powerful devices for image generation and can make efficient use of self-similarities in image compression, their generative power is incomparable with the well-known Iterated Function Systems (IFS), studied by Barnsley and others e.g. in [Barnsley 2000]. But there is a very natural way to extend WFA, such that they can simulate any IFS. Parametric Weighted Finite Automata (PWFA) are a multi-dimensional generalization of WFA introduced in [Albert and Kari 1999].

Definition 2. A Parametric Weighted Finite Automaton (PWFA for short) of dimension $d \in \mathbb{N}^+$ is a quintuple $Z = (Q, \Sigma, W, I, F)$, where

1. $Q = \{0, 1, \ldots, n-1\}, n \in \mathbb{N}^+$ is a finite non-empty set of states,
2. $\Sigma = \{0, \ldots, l-1\}, l \in \mathbb{N}^+$ is a finite non-empty alphabet,
3. $W = (W_0, \ldots, W_{l-1})$, where $W_i \in \mathcal{S}^{n \times n}$ is the matrix of weighted transitions for each input symbol $i \in \Sigma$,
4. $I = (I_0, I_1, \ldots, I_{d-1}), I_j \in \mathcal{S}^n$ are the initial distributions, where the I_j are the rows of the matrix I and
5. $F \in \mathcal{S}^n$ is the final distribution.

The overline operator in the following will denote the topological closure. Similar to the WFA case, the function (or word function) f computed by Z is defined as

$$f(w) = IW_{a_1} \ldots W_{a_k}F = I \prod_{i=1}^{k} W_{a_i} F \tag{10}$$

for each $w = a_1 \ldots a_k \in \Sigma^*$ and the set $S(Z)$ computed by Z is given by

$$S(Z) = \bigcap_{n=0}^{\infty} T_n(Z) \tag{11}$$

where

$$T_n(Z) = \overline{S_{\geq n}(Z)} \tag{12}$$

and

$$S_{\geq n} = \bigcup_{i=n}^{\infty} S_i(Z) \tag{13}$$

and

$$S_i(Z) = \{v | v = f(w) \text{ for some } w \in \Sigma^i\} \ . \tag{14}$$

We will use $T(Z) = T_0(Z)$ as an abbreviation. We will call a state s an initial state for dimension i, if and only if $I(i,s) \neq 0$ and an initial state of Z, if and only if there exists some dimension i such that s is an initial state for dimension i. We call a state s a final state, if and only if $F(s) \neq 0$.

The word function of every PWFA of dimension d over some semiring $\mathcal{S}$ can be simulated by a WFA over the semiring $\mathcal{S}^d$, where addition and multiplication are both performed component-wise for elements of $\mathcal{S}^d$ (cf. [Tischler 2007, Tischler 2008]). Still, PWFA are different from WFA in the following two important points:

1. Even if the semiring $\mathcal{S}$ is a field, $\mathcal{S}^d$ for $d \geq 2$ with component-wise addition and multiplication is never a field, for example the element $(0\ 0\ \ldots\ 0\ 1)^T$ has no inverse multiplicative element. Thus the well known minimization algorithms for WFA over fields cannot be used for PWFA of dimension greater than 1.

2. We mainly consider the computed set instead of the computed function.

The set of PWFA over each metric semiring can be divided into three categories. For one of these categories, we require the following definition.

Definition 3. Let (X, m) be a metric space. Furthermore, let 2^X denote the power set of X. Then we define

$$h^*(m)(A,B) = \begin{cases} 0 & \text{if } A = B = \emptyset \\ +\infty & \text{if } A \neq B = \emptyset \text{ or } B \neq A = \emptyset \\ \max\{\sup_{a \in A} \inf_{b \in B} m(a,b), & \text{otherwise} \\ \qquad \sup_{b \in B} \inf_{a \in A} m(a,b)\} & \end{cases} \tag{15}$$

as a function $h^* : 2^X \times 2^X \mapsto \overline{\mathbb{R}}_0^+$ for arbitrary sets $A, B \subseteq X$, where $\overline{\mathbb{R}} = \mathbb{R} \cup \{\pm\infty\}$.

The function h^* is a generalization of the Hausdorff metric. This is not a metric itself, but in some cases it is useful to determine the degree of similarity of non-compact sets.

Definition 4. Let Z denote a PWFA over the semiring $\mathcal{S}$ based on the metric space (X, m). We call Z

– strictly consistent, if $S(Z) = T(Z)$,

– limit consistent, if $\lim_{i \to \infty} h^*(m)(S(Z), T_i(Z)) = 0$, and

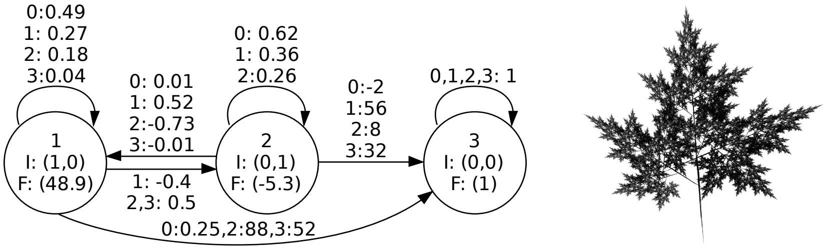

Figure 4: PWFA computing a fractal set (left) and the computed set interpreted as a 2D-bilevel-image (right)

– inconsistent, if it is not limit consistent.

The notion of strict consistency can be generalized for PWFA over non-metric semirings. Clearly, each strictly consistent PWFA is also limit consistent. It is an open problem, whether there are non-empty PWFA computable sets, which can not be represented by a limit consistent PWFA. It is also an open problem, whether there are sets that can be represented by limit consistent, but not by strictly consistent PWFA.

An example of a limit consistent but not strictly consistent PWFA is shown in Figure 4. This automaton is based on the fractal "maple leaf". It simulates an iterated function system (IFS) of dimension 2 over the real numbers (cf. [Barnsley 2000]). The automaton can be transformed into a strictly consistent automaton by choosing an appropriate final distribution representing a point on the attractor of the IFS. We conjecture that there is no effective algorithm to decide for given a PWFA representing an IFS whether it computes a point of its the attractor.

The family of PWFA computable sets over the real numbers has some interesting closure properties. It is closed under set union and regular restriction (cf. [Tischler 2004, Tischler 2005]). The set of sets which can be represented by limit consistent PWFA over $\mathbb{R}$ is closed under affine transformation and that which can be represented by strictly consistent PWFA over $\mathbb{R}$ is closed under iterated affine transformation (cf. [Tischler 2008]).

The PWFA we will construct to represent surfaces will be strictly consistent. When we transform a WFA representing an image to a PWFA by adding two dimensions computing the corresponding coordinate functions as given in equations 9, then this PWFA is limit consistent but in general not strictly consistent.

When we consider a WFA X, we have a fixed alphabet Σ_X and a function f_X computed by X. If we combine X with some other WFA Y to obtain a WFA Z satisfying certain properties (e.g. such that $f_Z = f_X + f_Y$), then we expect $\Sigma_Z = \Sigma_Y = \Sigma_X$. In the PWFA case, it is often useful to increase the alphabet size to obtain a certain result, for which we give an example in the construction below. We define the sum and product of two sets of elements of the same semimodule.

Definition 5. Let $\mathcal{S}$ be a semiring and d a positive natural number. Furthermore, let A and B be subsets of $\mathcal{S}^d$. We define the sum $A+B$ and product AB of A and B as

$$A + B = \{v | v = a + b \text{ for some } a \in A, b \in B\} \tag{16}$$

and

$$AB = \{v | v = a \cdot b \text{ for some } a \in A, b \in B\} \tag{17}$$

where $\cdot$ denotes the point-wise product.

For each semiring $\mathcal{S}$, each $d \in \mathbb{N}^+$ and each pair of sets $A, B \subseteq \mathcal{S}^d$ computable by strictly consistent PWFA the sets $A+B$ and AB are also PWFA computable.

Theorem 6. *Let $\mathcal{S}$ be a semiring based on a topological space such that the addition $+ : \mathcal{S} \times \mathcal{S} \mapsto \mathcal{S}$ and multiplication $\cdot : \mathcal{S} \times \mathcal{S} \mapsto \mathcal{S}$ in $\mathcal{S}$ are continuous functions and let $d \in \mathbb{N}^+$. Furthermore, let X and Y be strictly consistent PWFA of dimension d over $\mathcal{S}$. Then the sets $S(X) + S(Y)$ and $S(X)S(Y)$ are PWFA computable.*

Proof. Assume without loss of generality that $\Sigma_X = \Sigma_Y$. We say a PWFA is in initial normal form, if it has at most one initial state per dimension and in final normal form, if it has at most one final state. For each PWFA there are equivalent PWFA in initial or final normal form computing the same set (cf. [Tischler 2008]). These normal forms can be computed effectively. Without loss of generality assume that state 0 is the unique final state of the automaton X and that state i is the unique initial state for dimension i in the automaton Y. An automaton Z computing $S(Z) = S(X) + S(Y)$ can be obtained in two ways. The first construction works by doubling the number of labels. Let X' be the automaton obtained from X by doubling the number of labels and assigning the identity matrix to the labels $|\Sigma_X|, \ldots, 2|\Sigma_X| - 1$. Similarly let Y' be the automaton obtained from Y by doubling the number of labels, assigning the matrix W_{Y_i} to the label $|\Sigma_Y|+i$ for $i = 0, \ldots, |\Sigma_Y|-1$ and subsequently assigning the identity matrix to the labels 0 to $|\Sigma_Y| - 1$. Then the automaton computing the function $f_Z(w) = f_{X'}(w) + f_{Y'}(w)$ for all $w \in \Sigma_{X'}$ apparently computes the set $S(X) + S(Y)$. The second construction works by adding one label and $d+1$ states. Let the new symbol be denoted by ζ. Initially, let Z be a $d+1$ state PWFA of dimension d over $\mathcal{S}$ with $\Sigma_Z = \{0, \ldots, |\Sigma_X|\}$ such that

$$I_Z(i,j) = \begin{cases} 1 \text{ if } i+1=j \\ 0 \text{ otherwise} \end{cases} \tag{18}$$

for $i = 0, \ldots, d-1, j = 0, \ldots, d$,

$$F_Z = (F_X(0)\ f_Y(\epsilon)(0) \ldots f_Y(\epsilon)(d-1))^T \tag{19}$$

and W_{Z_i} is the identity matrix for $i \in \Sigma_X$ and the matrix which is 1 at position $(0,0)$ and zero otherwise for the symbol ζ. Then the automaton X is added to Z, where the transition matrix for the new symbol ζ is set to the null matrix. Subsequently the automaton Y is added to Z, where the transition matrix for ζ is set to W_{Y_0}. Finally we set $W_{Z_\zeta}(0_X, 0) = 1$ and $W_{Z_\zeta(i+1, i_Y)} = 1$ for $i = 0, \ldots, d-1$. Let $h : \Sigma_Z^* \mapsto \Sigma_X^*$ denote the morphism mapping the symbol ζ to 0 and each other symbol to itself. The automaton Z computes the function

$$f_Z(w) = \begin{cases} f_X(w) + f_Y(\epsilon) & \text{if } w \in \Sigma_X^* \\ f_X(\alpha) + f_Y(h(\beta)) & \text{if } w = \alpha\zeta\beta \text{ for } \alpha \in \Sigma_X^*, \beta \in \Sigma_Z^* \end{cases} \tag{20}$$

and thus $S(Z) = S(X) + S(Y)$.

For the generation of an automaton Z computing $S(Z) = S(X)S(Y)$ we generate d copies of X, which we call X_0 to X_{d-1} and change the copies by setting the final vectors F_{X_i} to

$$F_{X_i}(j) = \begin{cases} F_X(1) I_Y(i,i) F_Y(i) & \text{for } j = i \\ 0 & \text{otherwise} \end{cases} \tag{21}$$

for $j = 0, \ldots, |Q_X| - 1$ and the initial matrices I_{X_i} to

$$I_{X_i}(j) = \begin{cases} I_X(i) & \text{for } j = i \\ 0 & \text{otherwise} \end{cases} \tag{22}$$

for $i = 0, \ldots, d-1$. Then we erase the initial matrix of Y by setting $I(i) = 0$ for $i = 0, \ldots, d-1$. Let $\zeta = |\Sigma_X|$ be a new alphabet symbol. We set the transition matrix for the symbol ζ to the null matrix in X_i for $i = 0, \ldots, d-1$ and set $W_{Y_\zeta} = W_{Y_0}$. Then we construct the automaton Z by combining $X_0, \ldots, X_{d-1}$ and Y. Finally, we add new edges to Z by setting $W_{Z_\zeta}(1_{Q_{X_i}}, i_{Q_Y}) = F_X(1) I_Y(i,i)$ for $i = 0, \ldots, d-1$. The automaton Z works basically by simulating X and Y consecutively. It first simulates the basic behavior of X and after reading the new symbol ζ starts to simulate the basic behavior of Y. As above, let $h : \Sigma_Z^* \mapsto \Sigma_X^*$ denote the morphism mapping the symbol ζ to 0 and every other symbol to itself. The function computed by Z can be written as

$$f_Z(w) = \begin{cases} f_X(w) f_Y(\epsilon) & \text{if } w \in \Sigma_X^* \\ f_X(\alpha) f_Y(h(\beta)) & \text{if } w = \alpha\zeta\beta \text{ for } \alpha \in \Sigma_X^*, \beta \in \Sigma_Z^* \end{cases} \tag{23}$$

for each $w \in \Sigma_Z^*$, thus $S(Z) = S(X)S(Y)$.

Note that for computing the product $S(X)S(Y)$ of two sets $S(X)$ and $S(Y)$ computed by the strictly consistent PWFA X and Y we do not require the underlying semiring to be commutative.

3 PWFA representations for Bezier splines and patches

A Bezier curve $B^k : [0,1] \mapsto \mathbb{R}^d$ of order $k \in \mathbb{N}$ and dimension d is given by

$$B^k(t) = \sum_{i=0}^{k} B_i^k(t) P_i, \tag{24}$$

where $P_i \in \mathbb{R}^d, i = 0, \ldots, k$ are the control points defining the curve and the Bernstein polynomial B_i^k is given by

$$B_i^k(t) = \binom{k}{i} t^i (1-t)^{k-i} \tag{25}$$

for each $i, k \in \mathbb{N}, i \leq k$ [de Boor 1978, Maisonobe 2009].

Apparently each Bezier curve is a monadic polynomial defined on the real unit interval and thus can be represented using WFA and PWFA. A construction algorithm for WFA computing real monadic polynomials can be found in [Culik and Karhumäki 1994]. Assume that B_k is a Bezier curve of order k and dimension d. Then B_k can be represented either as a WFA over $\mathbb{R}^k$, where the product of two vectors is defined as the point-wise (Hadamard) product or as a PWFA of dimension d over $\mathbb{R}$. When we consider a Bezier curve of dimension greater than one, then we usually only consider the set of vectors produced for the whole unit interval and not which vector is produced by the curve at which point in the interval. Bezier curves can be used to model shapes. One important application of thereof is computer typography, where character outlines are defined by the splines. In applications a given curve is often modeled by a lengthy sequence of Bezier curves. This can easily be implemented via PWFA, since the set of "practically relevant PWFA functions" over the real numbers is closed under the set union operation and affine transformations. A PWFA for a curve from two Bezier splines is shown in Figure 5 together with its function. PWFA can also be used to represent other spline families (cf. [Tischler 2004, Tischler 2008]).

A Bezier patch $B^k : [0,1]^2 \mapsto \mathbb{R}^d$ of order k and dimension d for $k, d \in \mathbb{N}, d > 0$ is given by a matrix $Q \in \mathbb{R}^{d^{(k+1)\times(k+1)}}$ and the formula

$$B^k(u,v) = \sum_{i=0}^{k} \sum_{j=0}^{k} Q(i,j) B_i^k(u) B_j^k(v) \ . \tag{26}$$

Bezier patches can be used to model surfaces in $\mathbb{R}^3$. In applications we are mostly interested in Bezier patches of order 3 and dimension 3, which are called bicubic Bezier patches. PWFA computing multi-dimensional polynomials can easily be generated using PWFA computing delay functions of polynomials.

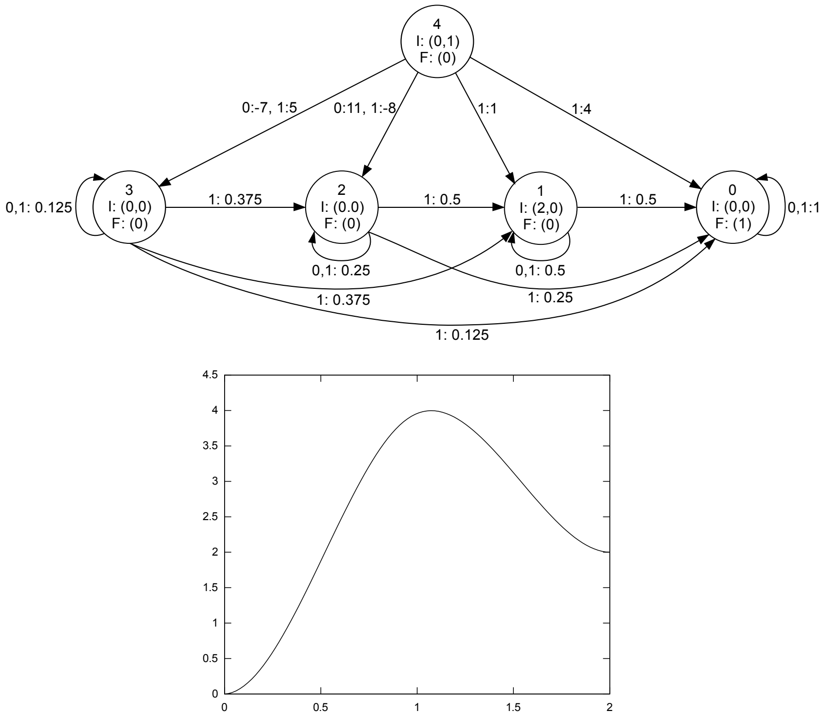

Figure 5: PWFA computing a curve from two Bezier splines

Definition 7. Let Σ be an alphabet and $k, m \in \mathbb{N}, m > 0, 1 \leq k \leq m$. Furthermore, let $w = a_1 \dots a_n \in \Sigma^*$. We call the word

$$o(k,m) = \begin{cases} a_k a_{k+m} \dots a_{k+m\lfloor \frac{n-k}{m} \rfloor} & \text{if } n \geq k \\ \epsilon & \text{otherwise} \end{cases} \tag{27}$$

the k projection of w modulo m.

Definition 8. Let Σ be an alphabet, S a set, $k, m \in \mathbb{N}, 1 \leq k \leq m$ and $f : \Sigma^* \mapsto S$ a function. We call the function $f(k,m)$ given by

$$f(k,m)(w) = f(o(k,m)(w)) \tag{28}$$

for each $w \in \Sigma^*$ the (k,m) delay function of f.

Delay functions of WFA computable functions are WFA computable (cf. [Tischler 2008]).

Let P be a WFA computing the real function $p(x) = x$, where the input word over the binary alphabet is interpreted using the bin function. Then we can define a real PWFA of dimension d representing a Bezier patch of order k and dimension d in the following way. If we insert the given elements of the coefficient matrix Q into formula 26, we see that B^k evaluates to a polynomial in two variables with coefficients from $\mathbb{R}^d$. W.l.o.g. we will only consider the one-dimensional case, $d = 1$. The generalization is straight-forward. Then we have a real polynomial with two variables with real coefficients,

$$B^k(u, v) = \sum_{i=0}^{k} \sum_{j=0}^{k} Q'(i, j) x^i y^j \tag{29}$$

for some matrix $Q' \in \mathbb{R}^{k+1 \times k+1}$. As WFA over the real numbers are closed under sum and product with scalars, we will assume w.l.o.g. that

$$B^k(u, v) = x^i y^j \tag{30}$$

for some $0 \leq i, j \leq k$. Assume that WFA X and Y computing the functions $f_{X^i}(x) = x^i$ and $f_{Y^j}(y) = y^j$. Then we can obtain a WFA X^iY^j computing the function $f_{X^iY^j}(w) = f_{X^i}(x_1 \ldots x_n) f_{Y^j}(y_1 \ldots y_n)$ for each $w = y_1x_1 \ldots y_nx_n$ by constructing a WFA for the Hadamard product of the functions computed by automata for the $(1, 2)$ delay function of f_{Y^j} and the $(2, 2)$ delay function of f_{X^i}. In practice it is sufficient to generate a WFA computing the function $x^k y^k$ to display a complete patch of order k, as this automaton contains state functions $x^i y^j$ for all $0 \leq i, j \leq k$, and set an appropriate initial distribution. Note that automata generated this way are unnecessarily large in general. This can be improved by the application of a minimization algorithm. The minimal automaton computing a component of a Bezier patch of order k has at most $(k + 1)^2$ states, which means 16 in the case of a bicubic Bezier patch. This bound in general is tight, there exist coefficient matrices Q such that the minimal automaton has $(k + 1)^2$ states.

It is based on the coefficient matrix

$$Q = \begin{pmatrix} (0\ 1\ 0) & (\kappa\ 1\ 0) & (1\ \kappa\ 0) & (1\ 0\ 0) \\ (0\ 1\ 0) & a & c & (1\ 0\ \kappa) \\ (0\ 1\ 0) & b & d & (\kappa\ 0\ 1) \\ (0\ 1\ 0) & (0\ 1\ \kappa) & (0\ \kappa\ 1) & (0\ 0\ 1) \end{pmatrix} \tag{31}$$

where $\kappa = \frac{4}{3}(\sqrt{2} - 1)$, and the vectors $a, b, c, d \in \mathbb{R}^3$ are given as

$$\begin{aligned} a &= (\kappa\ 1\ 0) + \sin(\pi/6)(0\ 0\ \kappa) \\ b &= (1\ \kappa\ 0) + \sin(\pi/3)(0\ 0\ \kappa) \\ c &= (0\ 1\ \kappa) + \sin(\pi/6)(\kappa\ 0\ 0) \\ d &= (0\ \kappa\ 1) + \sin(\pi/3)(\kappa\ 0\ 0) \end{aligned} \tag{32}$$

Also in the following equations we will leave $\sin(\pi/6)(= \sin(30°) = \frac{1}{2})$ and $\sin(\pi/3)(= \sin(60°) = \frac{\sqrt{3}}{2})$ as a reminder for the construction of the Bezier spline control points also at the angles of 30° and 60°.

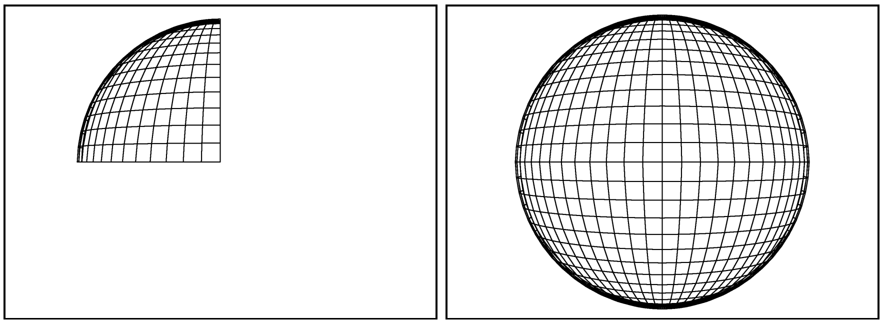

Figure 6: Single patch (left) and front part consisting of 4 patches (right) of an approximated unit sphere

The polynomials corresponding to the mappings of addresses (u, v) of the unit square to 3-dimensional addresses on the unit sphere are

$$\begin{aligned}
x(u,v) = {} & 3\kappa v + (3-6\kappa)v^2 + (3\kappa-2)v^3 + 9\kappa(\sin(\pi/6)-1)u^2v + \\
& 9(\kappa(\sin(\pi/3)+1)-1)u^2v^2 + \\
& (6(1-\kappa)+9\kappa(\sin(\pi/6)-\sin(\pi/3)))u^2v^3 + \\
& (\kappa(6-9\sin(\pi/6)))u^3v + \\
& (6-3\kappa(1+3\sin(\pi/3)))u^3v^2 + \\
& (3\kappa(1-3\sin(\pi/6)+3\sin(\pi/3))-4)u^3v^3, \\
y(u,v) = {} & 1 + 3(\kappa-1)v^2 + (2-3\kappa)v^3 \\
z(u,v) = {} & 9\sin(\pi/6)\kappa uv + 9\kappa(\sin(\pi/3)-2\sin(\pi/6))uv^2 + \\
& 3\kappa(3(\sin(\pi/6)-\sin(\pi/3))+1)uv^3 + \\
& 9(2\kappa(2\sin(\pi/6)-\sin(\pi/3)-1)+1)u^2v^2 + \\
& 3(\kappa(1+6\sin(\pi/3)-6\sin(\pi/6))-2)u^2v^3 + \\
& 3\kappa(2+3\sin(\pi/6))u^3v + \\
& 3(\kappa(4+3\sin(\pi/3)-6\sin(\pi/6))-2)u^3v^2 + \\
& (3\kappa(3\sin(\pi/6)-3\sin(\pi/3)-3)+4)u^3v^3 \ .
\end{aligned} \tag{33}$$

We use this patch as a building block for an approximated sphere, where a single patch represents one octant of the complete sphere. The union of eight affine transformations of the single octant patch then provides the sphere in total. Figure 6 shows the single patch with the front part of the sphere. In this view of the wire-frame model the back part has been omitted, as it would interfere with the front part in the rendering.

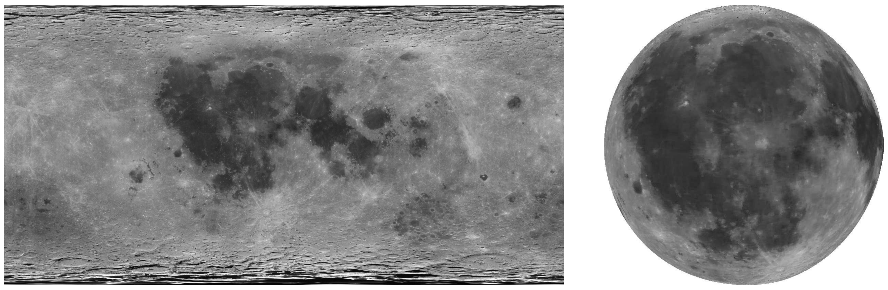

Figure 7: WFA-coded moon texture for an approximated sphere from eight PWFA spline patches

4 Textured spline patches

Textured spline patches can be represented efficiently by PWFA. We start by defining a surface by a spline patch in $\mathbb{R}^3$. The patch is defined by a function $s(u, v) : [0, 1]^2 \mapsto \mathbb{R}^3$, a function that maps locations from the unit square to points in $\mathbb{R}^3$. In a textured spline patch each point on the patch is assigned a vector of scalars, e.g. a single value from $[0, 1]$ describing a grayscale value or three values from $[0, 1]$ denoting a color described the intensities of the colors red, green and blue. Each point p on the patch in turn corresponds to a set of argument pairs (u, v).

Thus we can define a textured patch more formally as a function $t : [0, 1]^2 \mapsto \mathbb{R}^{3+c}$, where c is the number of color channels. As above, it is relevant only whether a point in the range of the function is generated, and not which argument pair actually generates it. Thus a textured patch can be represented as the set $\mathcal{T}$ given by

$$\mathcal{T} = \bigcup_{u\in[0,1],v\in[0,1]} \{t(u, v)\} \quad . \tag{34}$$

Consequently, we obtain a textured patch, by adding a WFA encoded image to the PWFA by increasing its dimension.

A more realistic example is shown in Figure 7. The texture of the moon surface from the website [Oera 2009] was WFA encoded and then applied to a set of eight bicubic Bezier patches forming an approximated unit sphere.

The representation of textured Bezier patches using PWFA is very succinct. The part of the automaton describing the form of the patch is based on a minimal WFA, the part representing the texture can be stored in a way competing with high performance image compression.

5 Conclusion

In this paper we have presented methods to generate PWFA for textured spline patches. The computed automata are succinct representations of the underlying graphical objects. For several important operations on those graphical objects there exist nice and effective closure properties. On the other hand, the list of open problems in PWFA construction includes the design of a general and effective inference algorithm for "small" PWFA from general textured or untextured data sets. These data sets would include textured 3D images of biological tissues and their surfaces as they occur in medical imaging for example in computer tomography or nuclear magnetic resonance spectroscopy.

References

[Aho et al. 2007] Aho, A. V., Lam, M. S., Sethi, R., Ullman, J. D.: "Compilers: Principles, Techniques, and Tools"; Addison-Wesley, 2007.

[Albert and Kari 1999] Albert, J., Kari, J.: "Parametric weighted finite automata and iterated function systems"; Proc. Fractals in Engineering, Delft, 1999, 248-255.

[Barnsley 2000] Barnsley, M. F.: "Fractals everywhere"; 2^{nd} ed., Morgan Kaufmann, 2000.

[Berstel and Nait Abdullah 1989] Berstel, J., Nait Abdullah, A.: "Quadtrees generated by finite automata", AFCET 61-62, (1989) 167-175.

[Berstel and Reutenauer 1988] Berstel, J., Reutenauer, C.: "Rational Series and Their Languages"; Springer-Verlag, 1988.

[de Boor 1978] de Boor, C.: "A practical guide to splines"; Springer-Verlag, 1978.

[Culik and Dube 1997] Culik II, K., Dube, S.: "Implementing Daubechies Wavelet Transform with Weighted Finite Automata"; Acta Inform.; 34, (1997) 5, 347-366.

[Culik and Karhumäki 1994] Culik II, K., Karhumäki, J.: "Finite automata computing real functions"; SIAM J. on Computing; 23, (1994) 4, 789-814.

[Culik and Kari 1993] Culik II, K., Kari, J.: "Image compression using weighted finite automata"; Computers & Graphics; 17, (1993) 3, 305-314.

[Culik and Kari 1994] Culik II, K., Kari, J.: "Image-Data Compression Using Edge-Optimizing Algorithm for WFA Inference"; Information Processing and Management; 30, (1994) 6, 829-838.

[Culik and Kari 1995] Culik II, K., Kari, J.: "Inference Algorithms for WFA and Image Compression"; Y. Fisher (ed.): Fractal Image Compression: Theory and Application; Springer-Verlag, 1995.

[Culik and Kari 1997] Culik II, K., Kari, J.: "Computational Fractal Geometry with WFA"; Acta Inform.; 34, (1997) 2, 151-166.

[Daubechies 1988] Daubechies, I.: "Orthonormal bases of compactly supported wavelets"; Comm. Pure and Applied Math.; 41, (1988) 909-996.

[Derencourt et al. 1992] Derencourt, D., Karhumäki, J., Latteux, M., Terlutte, A.: "On Computational Power of Weighted Finite Automata"; I. M. Havel, V. Koubek (eds.): Proc. MFCS '92; LNCS 629, Springer-Verlag, 1992, 236-245.

[Droste et al. 2006] Droste, M., Kari, J., Steinby, P.: "Observations on the Smoothness Properties of Real Functions Computed by Weighted Finite Automata"; Fundamenta Informaticae 73, (2006) 1,2, 99-106.

[Droste et al. 2009] Droste, M., Kuich, W., Vogler, H.: "Handbook of Weighted Automata", Monographs in Theoretical Computer Science. EATCS Series, Droste, M., Kuich, W., Vogler, H. (eds.), Springer-Verlag, 2009

[Eilenberg 1974] Eilenberg, S.: "Automata, Languages and Machines, Vol. A", Academic Press, New York, 1974.

[Hafner et al. 1998] Hafner, U., Albert, J., Frank, S., Unger, M.: "Weighted finite automata for video compression". IEEE J. on Selected Areas in Communications, 16 (1998) 108-119.

[Hafner 1999] Hafner, U.: "Low Bit-Rate Image and Video Coding with Weighted Finite Automata"; Ph.D. Thesis, University of Würzburg, Germany, 1999.

[JPEG2000 Core] ISO/IEC-Standard 15444-1: "Information technology - JPEG 2000 image coding system: Core coding system" `http://www.itu.int/rec/T-REC-T.800/en`

[Knuth et al. 1977] Knuth, D., Morris, J. H., Pratt, V.: "Fast pattern matching in strings"; SIAM Journal on Computing 6, 2 (1977) 323-350.

[Maisonobe 2009] Maisonobe, L.: "Drawing an elliptical arc using polylines, quadratic or cubic Bezier curves" at `http://www.spaceroots.org/documents/ellipse/index.htm`, 2009.

[Oera 2009] Øra, T.: "Texture maps of Earth and Planets" at `http://oera.net/How2/TextureMaps.htm`, 2009.

[Salomaa and Soittola 1978] Salomaa, A., Soittola, M.: "Automata-Theoretic Aspects of Formal Power Series" Springer-Verlag, Berlin, 1978.

[Schützenberger 1961] Schützenberger, M. P.: "On the Definition of a Family of Automata"; Information and Control; 4, (1961) 2-3, 245-270.

[Tischler 2004] Tischler, G.: "Parametric Weighted Finite Automata for Figure Drawing"; M. Domaratzki, A. Okhotin, K. Salomaa, S. Yu (eds.): Proc. Conf. Implementation and Application of Automata, CIAA 2004; LNCS 3317, Springer-Verlag, 2004, 259-268.

[Tischler 2005] Tischler, G.: "Properties and Applications of Parametric Weighted Finite Automata"; J. Automata, Languages and Combinatorics; 10, (2005) 2/3, 347-365.

[Tischler 2007] Tischler, G.: "On Computability and some Decision Problems of Parametric Weighted Finite Automata"; J. Automata, Languages and Combinatorics; 12, (2007) 4, 525-544.

[Tischler 2008] Tischler, G.: "Theory and Applications of Parametric Weighted Finite Automata"; Ph.D. Thesis, University of Würzburg, Germany, 2008.

[Tischler et al. 2005] Tischler, G., Albert, J., Kari, J.: "Parametric Weighted Finite Automata and Multidimensional Dyadic Wavelets"; Proc. Conf. Fractals in Engineering 2005; Tours, 2005.

[Wallace 1991] Wallace, G. K.: "The JPEG still picture compression standard"; Communications of the ACM; 34, (1991) 4, 30-44.

[Wood 1987] Wood, D.: "Theory of Computation"; Harper and Row, 1987.