In *Signal Processing*, special issue on image and video coding beyond standards, 82 (11): pp. 1581–1593, Nov. 2002.
`http://ltswww.epfl.ch/~dsanta`.

Coding of 3D virtual objects with NURBS

Diego Santa-Cruz and Touradj Ebrahimi

Signal Processing Laboratory
Swiss Federal Institute of Technology, CH-1015 Lausanne, Switzerland
E-mail: {Diego.SantaCruz,Touradj.Ebrahimi}@epfl.ch

Abstract

With the advancement of computer graphics in the recent years, an increasing number of pictures, video and 3D content is generated by synthesis processing rather than acquired with capture devices such as cameras or scanners. Several techniques have been developed for compression of discrete (i.e. piece-wise planar) 3D models, in the form of 3D polygonal meshes. However, no important attempt has been made to compress the smooth surfaces of artificially generated 3D models, that are most often represented as parametric surfaces, of which Non-Uniform Rational B-Spline (NURBS) is a popular form. This paper presents a method for compressing NURBS 3D models with a small and controllable loss. The scheme uses a differential pulse coded modulation (DPCM) coder with different predictors for knot values and control points, coupled with a uniform scalar quantizer, followed by a bitplane arithmetic entropy coder. The multiplicity of knots is preserved by the use of a multiplicity map. The rate-distortion characteristics of the proposed scheme are evaluated on various models. When compared to MPEG-4 [8,9] and Touma-Gotsman [19] compressed triangular meshes, the proposed scheme achieves more than five times better compression, for equivalent L^2 error and much better visual quality.

Key words: B-Spline, NURBS, compression, 3D model, coding, triangular mesh

1 Introduction

With recent progress in computing, algorithmics and telecommunications, 3D models are increasingly used in various multimedia applications. Examples include visualization, gaming, entertainment, and virtual reality. From a compression point of view, the use of 3D models has been extensively studied in the context of model based coding. In such an approach, when compressing a video captured by a camera, for instance, an analysis module attempts to model the scene under consideration as a set of (generally) 3D models. This

can be seen as an inverse projection problem. Once this task is successfully fulfilled, instead of coding the image sequence from the video, represented as a set of pixels and their motions, the parameters of the model extracted from the scene are coded. It is believed that if the analysis module is efficient enough, the total cost of coding (in a rate distortion sense) will be greatly reduced. The relatively poor performance and high complexity of currently available analysis methods (except for specific cases where *a priori* knowledge about the nature of the objects is available), has refrained a large deployment of coding techniques based on such an approach. Progress in computer graphics has changed this situation. In fact, nowadays, an increasing number of pictures, video and 3D content are generated by synthesis processing rather than coming from a capture device such as a camera or a scanner. This means that the underlying model in the synthesis stage can be used for their efficient coding without the need for a complex analysis module. This will also open the door to many applications enabled by other features offered by this approach.

On a parallel but related path, the way we consume audio-visual information is changing. As opposed to recent past and a large part of today's applications, interactivity is becoming a key element in the way we consume information. In the context of interest in this paper, this means that when coding visual information (an image or a video for instance), previously obvious considerations such as decision on sampling parameters are not so obvious anymore. To be more clear, let us provide an example. In a conventional digital video coding problem, the sampling (i.e. number of pixels in the image) mostly depends on the display resolution. Knowing the resolution of the display, there is no need to acquire and then to code a signal with sampling characteristics beyond those that the display is capable of reproducing. In an interactive environment, this is not true anymore. Even using a conventional display with, for instance, a resolution of 300×400 pixels, one could request to examine more closely (and therefore display) an object in a scene. This means that because of interactivity, the representation used to code the scene should allow the display of objects in a variety of resolutions, and ideally up to infinity. One way to resolve this problem would be by extensive over-sampling. But this approach is unrealistic and too expensive to implement in many situations. The alternative would be to use a resolution independent representation. Unfortunately, as far as 3D model coding is concerned, because of technological limitations in todays products, a sampling should be performed prior to display. Such a rendering includes discretization of 3D model objects. In particular, often continuous surfaces in objects are simplified into planar discrete portions, usually in form of connected triangular faces. This operation produces 3D meshes with other properties assigned to them such as colors, normals, textures, etc. Numerous techniques to efficiently represent and compress such meshes have been investigated [1,12,15,2,19] and even standards have been produced, such as MPEG-4 version 2 [8,9]. There has not been, so far, an important attempt to efficiently compress continuous 3D models, such as those obtained by para-

metric surfaces, which are among popular approaches to synthetise 3D objects. This paper proposes a solution to overcome this gap.

Many schemes for representing parametric surfaces exist, of which Non-Uniform Rational B-Spline (NURBS) patches [17] is one of the most popular. NURBS patches are a common tool for surface modeling, popular in CAD and virtual character generation, among others. NURBS have been already proposed for inclusion [6] in the Virtual Reality Modeling Language (VRML) standard [7], but in an uncompressed form. In this paper we propose a method to efficiently compress 3D models made of NURBS surface patches, with a small and controllable loss. In section 4 it is demonstrated that while the technique is relatively simple very good compression is obtained. In addition, the results show that the proposed scheme compares, from a rate-distortion point of view, very favorably to compressed triangular meshes, with the added benefit of retaining the resolution independence of the original parametric model.

2 NURBS patches

There is a variety of different schemes for representing NURBS surface patches. However the most common is *tensor product*. Other popular schemes are Bézier triangles, Coons and Rational Boundary Gregory (RBG) patches [14]. Due to the popularity and flexibility of tensor product patches, we restrict the discussion to those. In the following we will briefly review B-Spline functions, tensor product surfaces and NURBS patches. A detailed description of these concepts can be found in [18,5].

A tensor product surface $\mathbf{S}(u, v)$ is defined as a mapping from a region of $\mathbb{R}^2$ into Euclidean $\mathbb{E}^3$ space. Its general form can be expressed as:

$$\mathbf{S}(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} f_i(u) g_j(v) \mathbf{P}_{i,j} \qquad (u,v) \in [u_{min}, u_{max}] \times [v_{min}, v_{max}] \qquad (1)$$

where $\mathbf{P}_{i,j}$ is a set of $(n+1) \times (m+1)$ points in Euclidean $\mathbb{E}^3$ space, referred to as *control points*, $\{f_i(u)\}$ and $\{g_j(v)\}$ are two sets of univariate real valued functions, and u and v are the two independent parameters. The domain of definition in the parametric space is a rectangle and the topological arrangement of the control points is a rectangular grid.

For NURBS the basis functions are the B-Spline functions. Given $U = \{u_0, \ldots, u_r\}$, a non-decreasing sequence of $r+1$ real numbers (i.e. $u_i \leq u_{i+1}, 0 \leq i \leq r-1$), the ith B-spline function of pth degree (order $p+1$), denoted by $N_{i,p}(u)$, is

defined, recursively, as

$$N_{i,0}(u) = \begin{cases} 1 & \text{if } u_i \leq u < u_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{2a}$$

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \tag{2b}$$

where the quotient $0/0$ is defined to be zero. The u_i values are called *knots*, and U is the *knot vector*. Note that knots can have a *multiplicity* higher than one.

A polynomial B-Spline tensor product surface of degree p and q in the u and v directions, respectively, is thus defined by eqn. (1), where $f_i(u)$ is replaced by $N_{i,p}(u)$ and $g_j(u)$ is replaced by $N_{j,q}(v)$. The U and V knot vectors have $r+1$ and $s+1$ knots, respectively, where $r = n+p+1$ and $s = m+q+1$. The domain of definition becomes $(u,v) \in [u_{p+1}, u_{r-p-1}] \times [v_{q+1}, v_{s-q-1}]$. A NURBS or rational B-Spline surface is obtained by considering a polynomial tensor product surface in projective $\mathbb{P}^4$ space, which is then projected to affine $\mathbb{E}^3$ space. The resulting expression is thus

$$\mathbf{S}(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} R_{i,j}(u,v) \mathbf{P}_{i,j} \qquad \begin{matrix} u_{p+1} \leq u \leq u_{r-p-1} \\ v_{q+1} \leq v \leq v_{s-q-1} \end{matrix} \tag{3}$$

where $R_{i,j}(u,v)$ are the rational B-Spline functions of degree p and q, given by

$$R_{i,j}(u,v) = \frac{N_{i,p}(u) N_{j,q}(v) w_{i,j}}{\sum_{k=0}^{n} \sum_{l=0}^{m} N_{k,p}(u) N_{l,q}(v) w_{k,l}} \tag{4}$$

and $w_{i,j}$ is the homogenizing coordinate, or *weight*, of control point $\mathbf{P}_{i,j}$. That is the homogeneous coordinates of $\mathbf{P}_{i,j}$ are $(w_{i,j}\mathbf{P}_{i,j}, w_{i,j})$, with some abuse of notation. Note that, strictly speaking, a NURBS surface is not a tensor product surface once it has been projected to affine space.

The rational basis functions are well defined (i.e. the denominator is non-zero) over the domain $[u_{p+1}, u_{r-p-1}] \times [v_{q+1}, v_{s-q-1}]$ provided that all weights are non-zero and of the same sign. This is because of the non-negativity and partition of unity properties of the B-Spline functions $\{N_{i,p}(u)\}$ [18]. In the following it is assumed that $w_i > 0 \quad \forall\, i$, for all NURBS surfaces. In practice, this restriction is of very little importance, since all but the rarest design systems produce NURBS with positive-valued weights only. The effect of a weight $w_{i,j}$ is to move the surface $\mathbf{S}(u,v)$ closer or farther away from control point $\mathbf{P}_{i,j}$. The surface is farthest for $w_{i,j} \to 0$ and closest for $w_{i,j} \to \infty$, in which case $\mathbf{S}(u,v) \to \mathbf{P}_{i,j}$ for $(u,v) \in [u_i, u_{i+p+1}) \times [v_j, v_{j+q+1})$. Note that, in general, control points do not lie on the NURBS surface they define, although the control polygon approximates the surface.

NURBS surfaces posses several interesting properties. Those necessary to the discussion are outlined below. For a complete list as well as their proofs, the reader is referred to [18].

The parametric continuity of a NURBS surface $\mathbf{S}(u, v)$ is always C^∞, except at knot values. At a u knot of multiplicity k it is $C^{(p-k)}$ in the u direction. Analogously, at a v knot of multiplicity k it is $C^{(q-k)}$ in the v direction. The geometric (i.e. visual) continuity is however not entirely determined by parametric continuity. At a knot, it can be increased by placing the control points in a special way, or decreased by using multiple coincident control points.

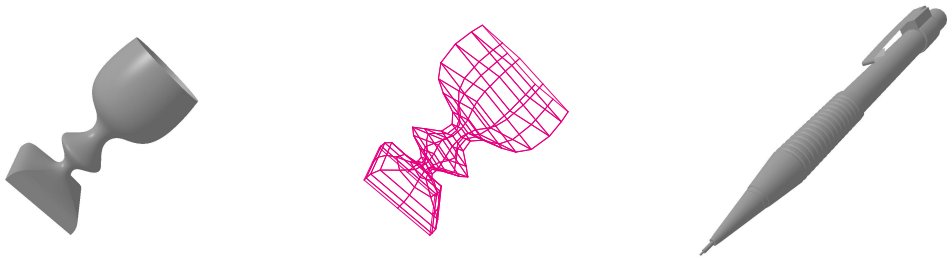
The first and last knots of any knot vector have no influence on a NURBS. In fact, given a knot vector U with $r + 1$ knots, it is possible to modify the values u_0 and u_r without affecting the NURBS, provided that the new knot vector is also a non-decreasing sequence of real values. Furthermore, the knot vectors can be normalized to the $[0, 1]$ range without modifying the shape of a NURBS surface, and without modifying its control points.

2.1 Typical knot vectors

Knot vectors define the basis functions of a NURBS surface. Besides being a non-decreasing sequence of real numbers, there are no additional constraints on knot vectors. However, several typical classes can be distinguished that are useful for our purposes:

- *Clamped* knot vectors, where the multiplicity of the first and last knots equals the order of the B-Spline function. That is, for the u direction, $U = \{\underbrace{a, \ldots, a}_{p+1}, u_{p+1}, \ldots, u_{r-p-1}, \underbrace{b, \ldots, b}_{p+1}\}$. For a NURBS surface which has clamped knot vectors in both directions, the corner control points coincide with the corners of the surface.
- *Uniform* knot vectors, where all the knots are uniformly spaced. That is $u_{i+1} - u_i \equiv c$ for $i = 0, \ldots, r - 1$.
- *Clamped uniform* knot vectors, where all the interior knots of a clamped knot vector are uniformly spaced.

Due to their special properties clamped and/or uniform knot vectors are used very often in designing NURBS surfaces. This fact will be exploited in the proposed coding scheme.



(a) (b) (c)

Fig. 1. NURBS models: (a) rendered surface of goblet, (b) control points and polygon of goblet, (c) rendered surface of pencil.

2.2 Example

In order to facilitate the understanding of the concepts introduced above, a brief example is provided. Figure 1a depicts the 3D model of a goblet, made of a single NURBS patch. This example clearly shows the flexibility of NURBS for representing surfaces: while the top of the goblet is a surface of revolution, the bottom features creases and a square base, yet the entire goblet is made up of a single NURBS patch. The patch is of degree two and three in the u and v directions, respectively. The 189 control points, which are also shown in figure 1b, can be organized in a matrix of 9 lines and 21 columns. Lines and columns correspond to the v and u directions, respectively. The v direction is parallel to the axis of the goblet, while the u is perpendicular to it. The knot vectors are as follows:

$$U = \{0, 0, 0, 0.25, 0.25, 0.5, 0.5, 0.75, 0.75, 1, 1, 1\}$$

$$V = \{0, 0, 0, 0, 0.05, 0.1, 0.124, 0.15, 0.2, 0.225, 0.25, 0.3, 0.325, 0.348, \\ 0.423, 0.494, 0.564, 0.585, 0.618, 0.684, 0.805, 1, 1, 1, 1\}$$

One can see that both knot vectors are clamped, and thus the corner of the patch and the corner control points coincide. The homogeneous coordinates of the control points in the last column of the control point matrix are

$$\begin{pmatrix} 1 \\ -0.24 \\ 0 \\ 1 \end{pmatrix}, \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ -0.24 \\ -0.24 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ -0.24 \\ 1 \end{pmatrix}, \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 0.24 \\ -0.24 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0.24 \\ 0 \\ 1 \end{pmatrix}, \\ \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 0.24 \\ 0.24 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 0.24 \\ 1 \end{pmatrix}, \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ -0.24 \\ 0.24 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ -0.24 \\ 0 \\ 1 \end{pmatrix}$$

These points correspond to the top edge of the goblet. It is easily seen that they lie on a square. However, the shape they describe is a circle. This is achieved by giving a weight of $1/\sqrt{2}$ to the points at the corner of the square. Note also that although the parametric continuity at the double knots of U

is C^0, the geometric continuity is C^∞, since it is a circle. This is due to the special placement of the control points. As is the case for the control points shown above, the first and last control points of each column in the control point matrix are the same. This coupled with the fact that the U knot vector is clamped, makes the surface closed in the u direction.

Figure 1c shows the much more complex pencil model. It is made of 11 rational and 6 non-rational NURBS patches and only clamped uniform knot vectors. The patches are of degree one and two, counting a total of 2514 control points. The largest patch has 203×9 (i.e. 1827) control points.

3 Coding scheme

In the following a powerful yet simple coding scheme for NURBS patches is presented. The knot vectors and control points are handled independently but in a similar manner. The basic building blocks are prediction, quantization and entropy coding, as depicted in figure 2. As shown in the figure, knots and control points are predicted differently. It is assumed, without loss of

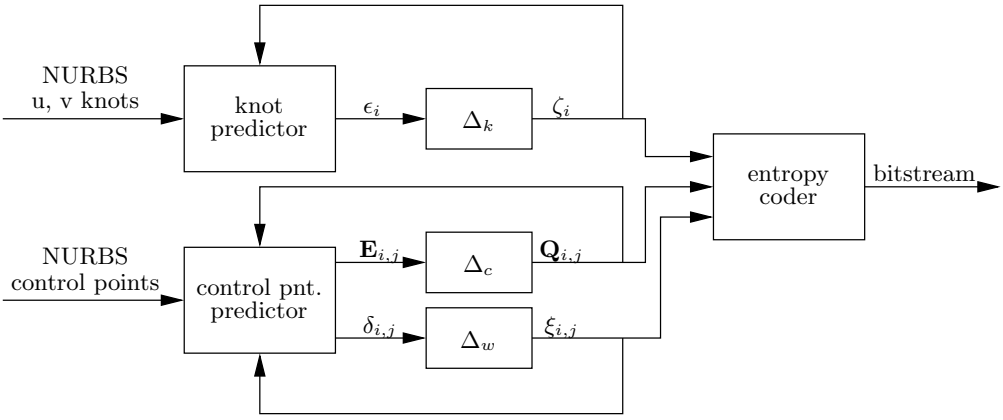


Fig. 2. Simplified block diagram of NURBS coder. Δ_k, Δ_c and Δ_w are three midrise uniform scalar quantizers.

generality, that all knot vectors are of the form $\{0, 0, u_2, \ldots, u_{r-2}, 1, 1\}$. Given that the first and last knots do not affect a NURBS and that a knot vector can be normalized, any knot vector that is not of the previous form can be cast into it. Likewise, it is also assumed, without loss of generality, that all the weights $w_{i,j}$ of the control points are in the interval $(0, 1]$. From eqn. (4) it is trivial to see that scaling all weights of a NURBS by a constant does not change the NURBS. Thus, dividing all weights by their maximum value yields the desired form.

3.1 Prediction and quantization

From the previous section, it is clear that both knot vectors and control points have some structure. This structure is exploited by the use of a predictive scheme, namely differential pulse coded modulation (DPCM), in order to reduce the redundancy of the coded data. Transform coding was also considered, however, both control points and knot vectors can often exhibit large discontinuities in their values which represent important features of the surface. A transform could introduce large distortions at these discontinuities. Furthermore, the number of samples in each patch is rather low and some transforms would have trouble exploiting the redundancy. For these two reasons, a predictive scheme seems more appropriate.

DPCM has been long studied in image compression [11], and optimal predictors and quantizers have been derived under various conditions for that application. However, the authors could not find evidence of it being applied to the problem at hand, where the nature and statistics of the data are fairly different and badly known. For NURBS, there is no clear relationship between the error on knots and/or control points, and the actual error on the surface itself. A conservative approach has thus been taken, where the maximum knot and control point error is guaranteed, given a quantization step size. This is achieved by the use of a uniform scalar quantizer.

3.1.1 Knot vectors

As previously mentioned, knot vectors define the shape of the B-Spline functions. In addition, the multiplicity of each knot value defines the continuity of the surface. It is thus important to preserve their multiplicity in the coding. Given a knot vector U, it is decomposed into the *break vector* $U' \equiv \{u'_i\}$ and the *multiplicity map* $U^m \equiv \{u^m_i\}$. The break vector contains the values of the knot vector, but where multiple knots appear only once, while the multiplicity map expresses the multiplicity of each knot, minus one. More formally

$$u^m_i = k_i - 1 \qquad i = 0 \dots r' \tag{5a}$$

$$u'_i = u_{i+\sum_{j=0}^{i} u^m_j} \qquad i = 0 \dots r' \tag{5b}$$

where k_i is the multiplicity of the break value u'_i in the knot vector U, and $r'+1$ is the number of elements in U' and U^m. The multiplicity map is losslessly entropy coded, as explained in section 3.2, without prior processing. For the break vector, DPCM and uniform scalar quantization are used prior to entropy coding. The order and number of knots is coded as overhead information in the bitstream header.

Break vectors are always a strictly increasing sequence. In addition, most

break vectors are uniform, or close to it. One can thus expect the difference between consecutive break values to remain almost constant, which is used as the DPCM predictor. The prediction error $\epsilon_i$to be coded can be thus expressed as

$$\epsilon_i = (u'_{i+1} - \hat{u}'_i) - (\hat{u}'_i - \hat{u}'_{i-1}) = u'_{i+1} - 2\hat{u}'_i + \hat{u}'_{i-1} \qquad i = 0, \ldots, r' - 2 \quad (6)$$

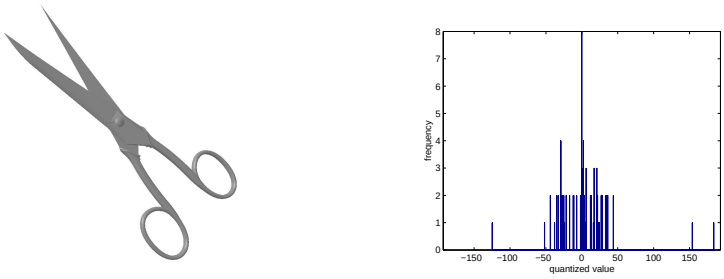
where the DPCM feedback loop is used to avoid the propagation of the quantization error. The decoded break values are denoted by $\hat{u}'_i$.

The prediction error is quantized with a midrise uniform scalar quantizer of step size Δ_k. The quantization index is thus $\zeta_i = \langle \epsilon_i / \Delta_k \rangle$, where $\langle \cdot \rangle$ denotes the rounding operator. The break values are thus decoded as

$$\hat{u}'_i = \hat{\epsilon}_{i-1} + 2\hat{u}'_{i-1} - \hat{u}'_{i-2} \qquad i = 1, \ldots, r' - 1 \quad (7)$$

where $\hat{\epsilon}_i = \zeta_i \Delta_k$ is the dequantized prediction error. The first and last break values are implicit: $\hat{u}'_0 = 0$ and $\hat{u}'_{r'} = 1$. In order to compute ϵ_0 the coder and decoder must agree in the value of $\hat{u}'_{-1}$. Following the fact that most break vectors are uniform the value is set to $\hat{u}'_{-1} = -1/r'$, so that ϵ_0 is minimized in such common cases. As for any DPCM coder with a uniform scalar quantizer, the coding error for break values is bounded by $\Delta_k/2$, as $|u'_i - \hat{u}'_i| = |\epsilon_{i-1} - \hat{\epsilon}_{i-1}| \leq \Delta_k/2$.

Figure 3 shows the histogram (calculated over 85 samples) of the DPCM prediction error, for the non-uniform break vectors of the scissors model. From the figure it is clear that the scheme produces a skewed distribution that is amenable to entropy coding.



(a) (b)

Fig. 3. (a) The scissors model, made of seven NURBS patches. (b) Histogram of quantized prediction error for the eight (out of 14) non-uniform break vectors of the scissors model.

Although effective, this prediction scheme cannot guarantee that the decoded break vector will be a sequence of strictly increasing values. If the quantization step size is too large, it can happen that $\hat{u}'_i > \hat{u}'_{i+1}$ for some i, leading to illegal break and knot vectors. Even the case of equality is not desired, since the multiplicity of some knots, and thus the continuity of the NURBS, would be

modified. An upper bound on Δ_k that guarantees proper coding is $\min\{u'_{i+1} - u'_i\}$, since that implies $u'_i + \Delta_k < u'_{i+1}$ and

$$\hat{u}'_i \leq u'_i + \Delta_k/2 < u'_{i+1} - \Delta_k/2 \leq \hat{u}'_{i+1} \Rightarrow \hat{u}'_i < \hat{u}'_{i+1} \text{ for all } i$$

However, this upper bound is not tight and larger values of Δ_k can still produce properly coded break vectors. In practice, the coder must check that the decoded break values form a strictly increasing sequence. This is fairly simple since a coder must always calculate the decoded break values in order to calculate the prediction. However, it can deem necessary the use of an iterative approach to find a suitable value for Δ_k, if the initial one is not small enough to produce a properly coded break vector. As it is shown in section 4, the bitrate necessary to properly code knot vectors is very low, and thus using a fine quantization step size is not a problem in itself.

3.1.2 Control points

NURBS control points can be either represented in projective space with homogeneous coordinates, or in affine space as regular 3D points with their associated weights. From the example of section 2.2, it is easy to see that the 3D points in affine space have more redundancy than the 4D points in projective space. This happens to be the case for most NURBS models. Based on this fact, the proposed scheme codes, for each control point, the affine coordinates and the weight, instead of the homogeneous coordinates. The only limitation in doing so is that it is not possible to handle NURBS with control points at infinity, without some special processing. However, control points at infinity are very rarely used, as their evaluation can be problematic.

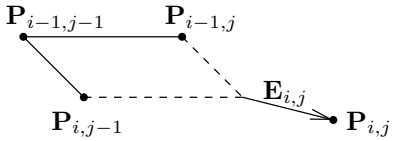


Fig. 4. Parallelogram prediction rule

The DPCM predictor used for the affine coordinates is the parallelogram rule which has been successfully used in the compression of vertex coordinates of 3D triangular meshes [19]. This rule predicts that given three known vertices, the fourth will form a parallelogram, as is shown in figure 4. The weights are also predicted in a similar fashion, but in 2D. Using vector notation for the control point coordinates and denoting as $\mathbf{E}_{i,j}$ the prediction error for coordinates and as $\delta_{i,j}$ the one for weights, one obtains:

$$\begin{aligned} \mathbf{E}_{i,j} &= \mathbf{P}_{i,j} - \hat{\mathbf{P}}_{i-1,j} - \hat{\mathbf{P}}_{i,j-1} + \hat{\mathbf{P}}_{i-1,j-1} \qquad i = 0, \ldots, n \\ \delta_{i,j} &= w_{i,j} - \hat{w}_{i-1,j} - \hat{w}_{i,j-1} + \hat{w}_{i-1,j-1} \qquad j = 0, \ldots, m \end{aligned} \tag{8}$$

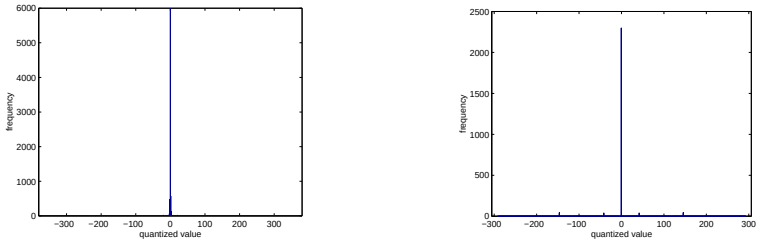
where the DPCM feedback loop has also been used. The decoded control point coordinates and weights are denoted by $\hat{\mathbf{P}}_{i,j}$ and $\hat{w}_{i,j}$, respectively.

The prediction errors are quantized using different midrise uniform scalar quantizers, of step size Δ_c and Δ_w. The prediction error $\mathbf{E}_i$ is however normalized by the decoded value $\hat{D}$, of the maximum side length D of the model's bounding box. The quantization indices are thus $\mathbf{Q}_{i,j} = \langle \mathbf{E}_{i,j}/(\hat{D}\Delta_c) \rangle$ and $\xi_{i,j} = \langle \delta_{i,j}/\Delta_w \rangle$, and the maximum error for the dequantized coordinate and weight values $\hat{D}\Delta_c/2$ and $\Delta_w/2$, respectively. The weights are, as previously mentioned, already normalized to the $(0, 1]$ range and require no further scaling.

The control point coordinates and weights are decoded as

$$\begin{aligned} \hat{\mathbf{P}}_{i,j} &= \hat{\mathbf{E}}_{i,j} + \hat{\mathbf{P}}_{i-1,j} + \hat{\mathbf{P}}_{i,j-1} - \hat{\mathbf{P}}_{i-1,j-1} \\ \hat{w}_{i,j} &= \hat{\delta}_{i,j} + \hat{w}_{i-1,j} + \hat{w}_{i,j-1} - \hat{w}_{i-1,j-1} \end{aligned} \tag{9}$$

where $\hat{\mathbf{E}}_{i,j} = \mathbf{Q}_{i,j}\hat{D}\Delta_c$ and $\hat{\delta}_{i,j} = \xi_{i,j}\Delta_w$ are the dequantized prediction errors. Along the topmost row and leftmost column the values $\{\hat{\mathbf{P}}_{-1,j}\}$ and $\{\hat{\mathbf{P}}_{i,-1}\}$ are considered equal to $\hat{\mathbf{P}}_{-1,-1}$. This effectively reduces the prediction to a zero order one from the left or top neighbor. Likewise, the values $\{\hat{w}_{-1,j}\}$ and $\{\hat{w}_{i,-1}\}$ are considered equal to $\hat{w}_{-1,-1}$. The value $\hat{\mathbf{P}}_{-1,-1}$ is set by the encoder to the middle of the model's bounding box and signaled in the header in exponent-mantissa representation. That is $\hat{\mathbf{P}}_{-1,-1} = 2^{\varepsilon_c}(1 + \mu_c/2^{M_c})$, where ε_c is the exponent and μ_c the mantissa. The number of bits M_c used to code the mantissa, and thus the precision of $\hat{\mathbf{P}}_{-1,-1}$, is determined so that the quantized prediction error $\mathbf{Q}_{0,0}$ is guaranteed to be within the allowed dynamic range (i.e. the maximum number of bits is not exceeded), and is of small magnitude. Note that all NURBS patches in a model use the same $\hat{\mathbf{P}}_{-1,-1}$ value. For weights the value $\hat{w}_{-1,-1}$ is set to one (i.e. $\hat{w}_{-1,-1} = 1$), since most often the control points have unit weight.



(a) (b)

Fig. 5. Histogram of quantized control point prediction error for all seventeen patches of the pencil model: (a) coordinate values $Q_{i,j}$; (b) weight values $\xi_{i,j}$. For the weights, only the truly rational patches are considered (eleven out of seventeen).

Figure 5 shows the histograms of the quantized coordinate and weight prediction error for the control points of the pencil model, calculated over 7542 and

2490 values, respectively. As it can be seen, the distribution of the values is very skewed, which is well adapted to an efficient entropy coding.

3.2 Entropy coding

The quantized values ζ_i, $\mathbf{Q}_{i,j}$ and $\xi_{i,j}$, as well as the knot multiplicity map U^m, are entropy coded using the MQ adaptive binary arithmetic coder, which is used in JPEG 2000 [3,10]. This arithmetic coder is very similar to the more widely known QM coder used in the JPEG [16] standard.

A simple, yet effective, bitplane arithmetic coder is used to compress the quantized values of break vectors and control points. The values are coded in their sign-magnitude representation. The same coder is used for break values and control point coordinates and weights. In order to ensure that no overflow can occur, $N_k = 1 - \lfloor \log_2 \Delta_k \rfloor$ magnitude bits are necessary to code a break value, where $\lfloor \cdot \rfloor$ denotes the floor operator. This limit is derived by considering the maximum value of the prediction error, with respect to the quantization step size. Similarly, $N_c = 2 - \lfloor \log_2 \Delta_c \rfloor$ and $N_w = 2 - \lfloor \log_2 \Delta_w \rfloor$ magnitude bits are necessary to code control point coordinates and weights, respectively.

Many knot vectors are uniform. Since uniform knot vectors, clamped or not, can be determined by the number of their values, it is not necessary to code their break vector nor their multiplicity map. Similarly, uniform break vectors only require the coding of the multiplicity map. Therefore, clamped and uniform knot vector flags are coded, followed by an eventual uniform break vector flag. The uniform knot vector flag is coded with a non-adaptive context with uniform distribution. The clamped and uniform break vector flags are coded with contexts CKF and UBF, respectively.

Coding of quantized break values proceeds from the most significant magnitude bitplane, $N_k - 1$, to the least significant, 0. For typical break vectors, $\max\{\zeta_i\}$ is often much smaller than $2^{N_k} - 1$. Therefore, many of the leading most significant bitplanes of $\{\zeta_i\}$ will be all zero. Let z_k be their number. Instead of coding all these zero bits separately, z_k is signaled by coding, with the ZBP context, z_k 0 followed by one 1. After that, bitplanes $N_k - 1 - z_k$ to 0 follow, where each ζ_i is coded separately. For each ζ_i, the leading zero most significant bits, as well as the leading non-zero one, are coded with the LZERO context. The remaining magnitude bits, called magnitude refinement bits, are coded with the MREF context. The sign is coded separately with the SIGN context.

The entropy coding for control point coordinates is analogous, but with a separate set of contexts. However, the prediction of the topmost row and leftmost column of control points is often not as good as for the other positions,

since it is based on the value of only one neighbor. This decreases the number of most significant all-zero bitplanes z_c and thus increases the number of bitplanes to be coded. To avoid this, the values $\{\mathbf{Q}_{0,j}\}_{j\geq 0}$, and $\{\mathbf{Q}_{i,0}\}_{i\geq 0}$, are ignored when calculating z_c. A value z'_c is calculated for $\{\mathbf{Q}_{0,j}\}_{j>0}$ and $\{\mathbf{Q}_{i,0}\}_{i>0}$, but still ignoring $\mathbf{Q}_{0,0}$ for very much the same reason. The values are then coded in the same way as for the quantized break values, except that N_c bits are coded for $\mathbf{Q}_{0,0}$, $N_c - 1 - z'_c$ for $\{\mathbf{Q}_{0,j}\}_{j>0}$ and $\{\mathbf{Q}_{i,0}\}_{i>0}$, and $N_c - 1 - z_c$ for the others. Different LZERO contexts are used for each set and different ZBP contexts are used to signal z'_c and $z_c - z'_c$. The entropy coding of control point weights is identical to the one of coordinates.

In the case where a knot vector is non-uniform, its multiplicity map U^m needs to be coded. The entropy coder in this case is similar to that used for DPCM coding in JPEG [16], which is a modified version of the method devised by Langdon [13]. The algorithm to code each u_i^m value can be summarized as follows:

if $u_i^m = 0$ **then**
  Code a one with the MZERO context
else
  Code a zero with the MZERO context
  $m_i \leftarrow \lceil \log_2(u_i^m) \rceil$
  for $j = 0$ to $m_i - 1$ **do**
    Code a one with the MEXP+j context
  end for
  Code a zero with the MEXP+m_i context
  if $m_i \geq 2$ **then**
    Code bits $m_i - 2$ to 0 of $u_i^m - 1 - 2^{m_i}$ with the MMAG+$m_i - 1$ context.
  end if
end if

The basic idea is to choose the statistics based on the magnitude of the values. First the $\log_2$ bin of $u_i^m - 1$ is coded using the $\{\text{MEXP}+j\}$ contexts. Then the magnitude refinement bits are coded using the statistics corresponding to the bin, that is with the MMAG+$m_i - 1$ context. The case $u_i^m = 0$ is treated separately. Note that for clamped knot vectors $u_0^m - (p+1)$ is coded instead of u_0^m, where $p+1$ is the order of the basis function. Likewise, $u_{r'}^m - (p+1)$ is coded instead of $u_{r'}^m$.

4 Experimental results

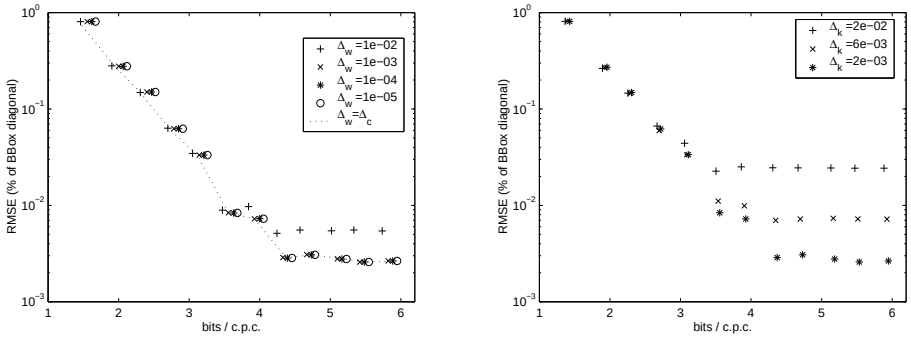
The rate distortion performance of the proposed NURBS coding scheme has been evaluated. The distortion between the coded and original surfaces is

measured by means of the Hausdorff distance. Let $d(p, S')$ be the Euclidean distance from a point p on surface S to the closest point on surface S' (i.e. the Hausdorff distance). The one-sided L^2 distance between S and S', $d(S, S')$, is given by

$$d(S, S') = \left(\frac{1}{area(S)} \int_{p \in S} d(p, S')^2 \mathrm{d}p \right)^{1/2}$$

This distance is clearly not symmetric. The two-sided distance is defined as $\max\{d(S, S'), d(S', S)\}$, which is symmetric. Evaluating this two-sided L^2 distance directly on the NURBS surface is an extremely difficult task. However, NURBS models can be tessellated as very fine triangular meshes, and the distance between these is measured by the Metro tool [4]. In the following, the L^2 error is expressed relative to the length of the model's axis aligned bounding box diagonal, and the rate as the number of bits per control point coordinate (bits/c.p.c.).

In order to vary the bitrate of a coded NURBS, three quantization step sizes (Δ_k, Δ_c and Δ_w) can be adjusted independently. It is clear that, in general, reducing one quantization step size, without modifying the others, will increase the bitrate and reduce the distortion. However, the optimum setting for these three parameters that minimizes the distortion for a given bitrate is not well determined. Nevertheless, as is shown below, some simple rules can be applied to find a semi-optimal trade-off between the different parameters.

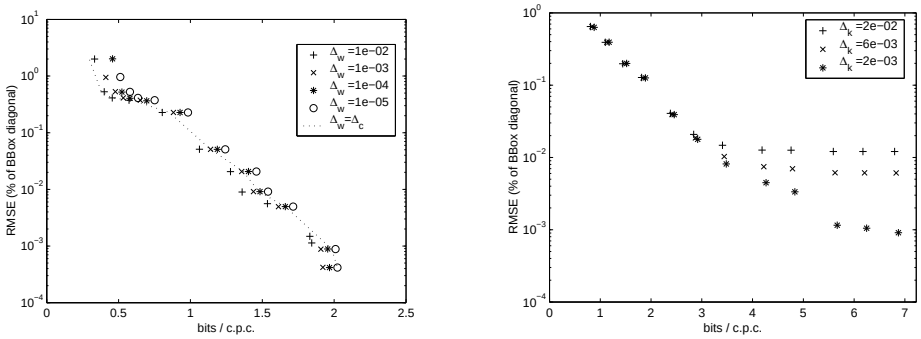


(a) (b)

Fig. 6. Rate-distortion curves for the goblet model. (a) Curves for fixed $\Delta_k = 2 \cdot 10^{-3}$ and different Δ_w, for which Δ_c is varied to obtain different bitrates. (b) Curves for $\Delta_w = \Delta_c$, and various Δ_k.

Figure 6a shows various rate-distortion curves for the goblet model, where the knot quantization step size Δ_k is fixed to some small value. The dotted line shows the rate-distortion curve when $\Delta_w = \Delta_c$. As it can be seen, $\Delta_w = \Delta_c$ is close to optimal for all bitrates, as opposed to some fixed Δ_w value. Furthermore, decreasing Δ_w below some value (in this case $2 \cdot 10^{-3}$) does not decrease the distortion by any noticeable amount, at any bitrate, although it decreases the coding efficiency. Finally, one can also observe that, beyond some bitrate (in this case 4 bits/c.p.c.), almost no reduction in distortion

occurs. That is, the system "saturates". Figure 6b shows the rate-distortion curves for various values of Δ_k, where $\Delta_w = \Delta_c$. One can observe that the aforementioned "saturation" is, as it could be expected, due to the value of Δ_k: beyond some bitrate, it is necessary to decrease Δ_k to obtain a noticeable reduction in distortion. Finally, one can also see that the increase in bitrate incurred by reducing Δ_k tenfold is fairly small. This can be explained by the fact that the number of knot values is normally small when compared to the number of control point coordinates and weights and that uniform knot or break vectors are efficiently coded. Therefore, using a value of Δ_k that is smaller than optimal will not adversely affect the compression efficiency.



(a) (b)

Fig. 7. (a) Pencil model's rate-distortion curves for fixed $\Delta_k = 2 \cdot 10^{-3}$ and different Δ_w, for which Δ_c is varied to obtain different bitrates. (b) Scissors model's rate-distortion curves for $\Delta_w = \Delta_c$, and various Δ_k.

Figure 7a shows the same results as figure 6a, but for the pencil model. In this case, weight coding represents a significant amount of the total bitrate and thus the trade off between Δ_c and Δ_w is more involved. The $\Delta_w = \Delta_c$ setting is not near-optimal anymore, although it still remains a reasonable simplification. For the pencil model the setting of Δ_k has no influence on the coding efficiency, because all knot vectors are uniform.

Figure 7b shows results for the scissors model. Similar remarks apply. However, for this model, the setting of Δ_w has a very small effect on the coding efficiency. This is because there is only one truly rational patch and thus weight coding represents only a small fraction of the overall bitrate.

As a basis for comparison, table 1 shows the file sizes of the original NURBS models, in ASCII and binary form, compressed with the popular general purpose `gzip` compressor. By comparing these numbers to the results shown in figs. 6 and 7, it is clearly seen that the proposed coding scheme is very advantageous, while remaining simple. For an L^2 error of 1 in 10^4, around three to four times better compression than `gzip`'ed binary is achieved. For an error of 1 in 10^3 that factor increases to 3.8 or even 6.6, depending on the model. Figure 8 shows the rendered decoded models, where no visual differences are visible with respect to the original ones. The proposed scheme achieves, for

	ASCII	ASCII	binary	binary
	bits/c.p.c.	bytes	bits/c.p.c.	bytes
goblet	13.53	1279	10.11	956
pencil	7.42	9303	6.23	7814
scissors	15.45	5961	13.15	5075

Table 1
Compressed size of original models, in ASCII and binary (32 bit floating point). Compressed with `gzip`, maximum level.

visually lossless results, a compression factor four or more times larger than the lossless compression obtained by `gzip`'ing the binary files.

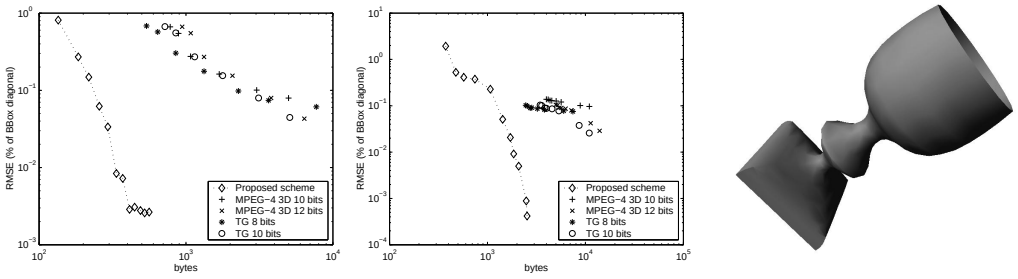
(a) (b) (c)

Fig. 8. Coded NURBS models: (a) goblet, 2.3 bits/c.p.c., L^2 error $14.8 \cdot 10^{-4}$; (b) pencil, 1.0 bits/c.p.c., L^2 error $5.1 \cdot 10^{-4}$; (c) scissors 2.4 bits/c.p.c., L^2 error $3.9 \cdot 10^{-4}$;

Finally, figure 9 compares the proposed NURBS coding scheme with compressed polygonal meshes obtained by tessellation of the goblet and pencil models. The mesh compression techniques used are those of MPEG-4 [9], namely the 3DMC reference software of Feb. 21, 2001, and Touma-Gotsman [19]. It is clearly seen that compressed NURBS are much more compact than polygonal meshes, even in the case of coarse ones. The proposed scheme, besides providing more than 5 times better compression, provides much better visual quality for comparable L^2 errors. In fact, surface normals are implicit in NURBS, whereas they must be estimated when rendering the compressed polygonal meshes. MPEG-4 allows to code the "true" normals of a model, but that increases even more the difference in coding gain with respect to the proposed scheme.

5 Conclusions and future work

In this paper we propose a method for lossy compression of NURBS models generated by 3D model synthesis processing, such as CAD and virtual character generation systems. The method employs a DPCM coder with ad-hoc



(a) (b) (c)

Fig. 9. Rate-distortion behavior of compressed polygonal meshes and the proposed NURBS coding scheme for the (a) goblet and (b) pencil models. (c) Visual results for Touma-Gotsman on the goblet model with 612 vertices and 1192 triangles, compressed size is 1328 bytes and the L^2 error is $17.6 \cdot 10^{-4}$ with respect to the original NURBS model.

predictors and a uniform scalar quantizer, followed by bitplane arithmetic entropy coding. The use of a uniform scalar quantizer guarantees a maximum bound for the error on the various NURBS parameters. The rate-distortion characteristics of the method have been evaluated on various models, and compared to the lossless compression ratios obtained on the original models. The results show that the proposed method is efficient while retaining visually lossless characteristics. In addition, a comparison has been made with compressed triangular meshes derived from the NURBS models, which clearly shows the advantage of using compressed NURBS instead of triangular meshes.

Although the presented method proves to be effective, two main subjects deserve further work. First of all, a rate-allocation scheme that takes into account the surface distortion, instead of the distortion of the NURBS parameters themselves, could improve the rate-distortion performance. Second, and most important, the connectivity relationships between the various NURBS patches of a model need to be taken into account to make a robust coding system. Within a model, the various NURBS patches are often adjacent to one another, making a continuous surface. However, the distortion introduced by the coding process can make the (originally) adjacent patches not adjacent anymore. As a consequence, visually visible cracks appear in the decoded model. Note that using triangular meshes derived from the model can suffer from the same problem.

Acknowledgements

The NURBS models used to evaluate the proposed scheme were created with the Alpha_1 geometric modeling system at the Computer Science Department, University of Utah.

References

[1] Pierre Alliez and Mathieu Desbrun. Valence-driven connectivity encoding for 3D meshes. *Computer graphics forum*, 20(3):480–489, 2001. Presented at EUROGRAPHICS 2001, 4-7 Sept. 2001, Manchester, UK.

[2] Jin Soo Choi, Yong Han Kim, Ho-Jang Lee, In-Sung Park, Myoung Ho Lee, and Chieteuk Ahn. Geometry compression of 3-D mesh models using predictive two-stage quantization. *IEEE Trans. on Circuits and Systems for Video Technology*, 10(2):312 – 322, March 2000.

[3] Charilaos Christopoulos, Athanassios Skodras, and Touradj Ebrahimi. The JPEG2000 still image coding system: an overview. *IEEE Trans. on Consumer Electronics*, 46(4):1103–1127, November 2000.

[4] P. Cignoni, C. Rocchini, and R. Scopigno. Metro: measuring error on simplified surfaces. *Computer Graphics Forum*, 17(2):167–174, June 1998.

[5] Gerald E. Farin. *NURBS: from projective geometry to practical use*. A K Peters, Ltd., Natick, Massachusetts, 2nd edition, 1999.

[6] Holger Grahn, Thomas Volk, and Hans J. Wolters. Nurbs in VRML. In *Proc. of the Web3D-VRML 2000 fifth symposium on Virtual reality modeling language*, pages 35–43, Monterey, CA, USA, February 2000. ACM Press.

[7] ISO/IEC. *ISO/IEC 14772-1:1998 Information technology — Computer graphics and image processing — The Virtual Reality Modeling Language (VRML) — Part 1: Functional specification and UTF-8 encoding*, June 1998.

[8] ISO/IEC. *ISO/IEC 14496-2:1999: Information technology — Coding of audio-visual objects — Part 2: Visual*, December 1999.

[9] ISO/IEC. *ISO/IEC 14496-2:1999/Amd 1:2000: Visual extensions*, August 2000.

[10] ISO/IEC JTC 1/SC 29/WG 1. *ISO/IEC FDIS 15444-1: Information technology — JPEG 2000 image coding system: Core coding system [WG 1 N 1890]*, September 2000.

[11] Anil K. Jain. *Fundamentals of Digital Image Processing*. Prentice Hall, Englewood Cliffs, New Jersey, 1989.

[12] Andrei Khodakovsky, Peter Schröder, and Wim Sweldens. Progressive geometry compression. In *Proc. of the International Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 2000*, pages 271–278, New Orleans, LA, USA, July 2000.

[13] Glen G. Langdon. Compression of multilevel signals. Patent Nr. 4,749,983, United States, June 1988.

[14] Duncan Marsh. *Applied Geometry for Computer Graphics and CAD*. Springer-Verlag, London, 1999.

In *Signal Processing*, special issue on image and video coding beyond standards, 82 (11): pp. 1581–1593, Nov. 2002.
`http://ltswww.epfl.ch/~dsanta`.

[15] Renato Pajarola and Jarek Rossignac. Compressed progressive meshes. *IEEE Trans. on Visualization and Computer Graphics*, 6(1):79–93, January-March 2000.

[16] William B. Pennebaker and Joan L. Mitchell. *JPEG: Still Image Data Compression Standard*. Van Nostrand Reinhold, New York, 1992.

[17] Les Piegl. On nurbs: a survey. *IEEE Computer Graphics and Applications*, 11(1):55–71, January 1991.

[18] Les Piegl and Wayne Tiller. *The NURBS Book*. Springer-Verlag, Berlin, Germany, 2nd edition, 1997.

[19] Consta Touma and Craig Gotsman. Triangle mesh compression. In *Proc. of the 24th Conference on Graphics Interface (GI-98)*, pages 26–34, San Francisco, CA, USA, June 1998. Morgan Kaufmann.