An Automated Video Data Compression Algorithm by Cardinal Spline Fitting

M A Khan and Yoshio Ohno
Graduate School of Science and Technology, Keio University
E-mail:[murtaza,ohno]@on.cs.keio.ac.jp

Abstract

This paper proposes an algorithm for compression of color and gray-level video data using Cardinal spline. Lossy data compression method is proposed using Cardinal spline least square fitting to temporal variations of pixel values. Upper bound pixel level fitting accuracy is achieved by break-and-fit criteria. The method facilitates fully automated, simple and efficient compression/decompression video data for both naturally recorded and synthetically created videos. A compact representation of output bitstream and bitstream parsing algorithm are also described.

1 Introduction

Digital video data consists of sequence of frames (images). Each frame consists of 2–*D* array of pixels. 3–*D* *RGB* or 1–*D* *luminance* values are associated with each pixel. Value of a pixel in a frame can be considered as a point in Euclidean space R^3 or R^1. Therefore if a video consists of a sequence of n frames then for each pixel we have a set of values $\{p_1, p_2, \ldots, p_n\}$, i.e., $p_j = (R_j, G_j, B_j)$ for *RGB* or $p_j = I_j$ for *luminance*, where $1 \leq j \leq n$. Figure 1 shows *RGB* variation of a pixel in 80 frames of a video.

Video data contains spatial and temporal redundancy.

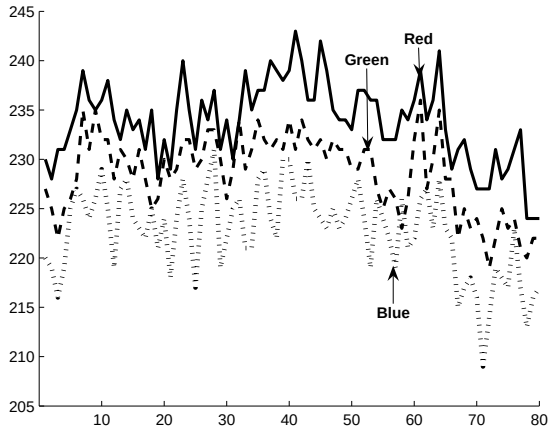

Figure 1: *RGB* temporal variation of a pixel in 80 frames of a video.

Spatial redundancy is removed by image compression techniques while temporal redundancy is removed by techniques like motion compensation. In our proposed video data approximation method focus is on reducing temporal redundancy of data by approximating it using cubic Cardinal spline. Due to large size of video data it is also desirable that compression process is automated. In our proposed scheme only initially user has to set the few parameters then rest of the compression and decompression steps are fully automated.

Organization of the rest of the paper is as follows: Related work is discussed in Sect. 2. Cardinal spline interpolation and least square fitting are explained in Sect. 3 and Sect. 4 respectively. Section 5 is the most important section and it describes the details of fitting strategy of the proposed algorithm. Section 6 describes the method to construct the bitstream of output data and an algorithm to parse the bitstream. Selected results are presented in Sect. 7. Section 8 discusses about algorithm and rational to choose Cardinal spline. Final concluding remarks are in Sect. 9.

2 Related Work

An extensive work has been done on curve fitting [2, 3, 9] et al. But almost in all techniques fitting is for non-video data. Cardinal spline interpolation is investigated by many authors [5, 1, 7] but approximation via least square Cardinal spline fitting is not explored. The proposed method explores the power of Cardinal spline fitting in the domain of video data. Conventional

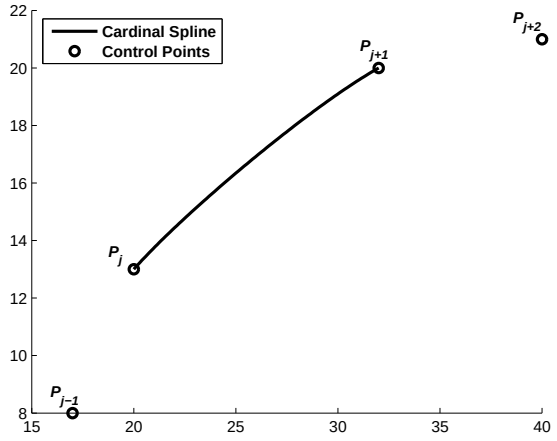

Figure 2: The j^{th} Cardinal spline segment, $T = 0$.

methods of temporal video compression are based on block matching algorithms (BMAs) [4, 10, 6]. BMA works by taking group of pixels together as a block and finds the matching block with given tolerance of error. BMA works at block level and may cause blocking artifacts. The proposed method works at pixel level, therefore more precise control of accuracy at pixel level and immunity from blocking artifacts is achieved. A video coding scheme for 3–*D* television broadcasting is proposed by [8]. Our work is also related to compression of video data but in contrast to [8] we did not focus on 3–*D* systems.

3 Cardinal Spline

Cubic Cardinal spline is a C^1 continuous curve. Cardinal spline interpolates piecewise cubics with specified endpoint tangents for each segment. A Cardinal spline segment, as shown in Fig. 2, is defined by four control points, i.e., P_{j-1}, P_j, P_{j+1} and P_{j+2}. The j^{th} segment of Cardinal spline interpolates between two *middle control points*, i.e., P_j and P_{j+1}. The *end control points*, i.e., P_{j-1} and P_{j+2} are used to calculate the tangent of P_j and P_{j+1}. Equations for boundary conditions of j^{th} segment is written as:

$$P'_j = \frac{1}{2}(1-T)\left(P_{j+1}-P_{j-1}\right), \quad (1)$$

$$P'_{j+1} = \frac{1}{2}(1-T)\left(P_{j+2}-P_j\right), \quad (2)$$

where parameter T is *Tension* and it controls looseness/tightness of spline.

For l joined segments, there are $2(l-1)$ conditions for continuity of functions and $2(l-1)$ conditions for continuity of slopes. Finally the equation of Cardinal spline for j^{th} segment is written as follows:

$$\begin{aligned} Q(t_i) =& (-st_i^3 + 2st_i^2 - st_i)P_{j-1} \\ &+ [(2-s)t_i^3 + (s-3)t_i^2 + 1]P_j \\ &+ [(s-2)t_i^3 + (3-2s)t_i^2 + st_i]P_{j+1} \\ &+ (st_i^3 - st_i^2)P_{j+2}, \end{aligned} \quad (3)$$

where t_i is parameter of interpolation , $0 \le t_i \le 1$, and s is related to *Tension* by $s = \frac{(1-T)}{2}$. In order to generate n points between P_j and P_{j+1} inclusive, the parameter t_i is divided into $(n-1)$ intervals between 0 and 1 inclusive, and $Q(t_i)$ is evaluated (interpolated) at n values of t_i.

4 Cardinal Spline Least Square Solution

For video data the first and last control points of Cardinal spline are pixel values in two specified adjacent keyframes. But the value of T must be known. In order to find the best value of s (or T) that minimizes the square distance between original data and fitted data, Eq. (3) can be solved using least square method as follows:

$$X = \sum_{i=1}^{n} [p_i - Q(t_i)]^2. \quad (4)$$

$$\frac{\partial X}{\partial s} = 0. \quad (5)$$

Solving Eq. (5) for s gives:

$$s = \frac{\sum_{i=1}^{n} D_i p_i - \sum_{i=1}^{n} D_i E_i}{\sum_{i=1}^{n} D_i^2}, \quad (6)$$

where

$$A_i = t_i^3 - 2t_i^3 + t_i, \quad (7)$$

$$B_i = t_i^3 - t_i^2, \quad (8)$$

$$C_i = 2t_i^3 - 3t_i^2, \quad (9)$$

$$D_i = \left(P_{j+1} - P_{j-1}\right)A_i + \left(P_{j+2} - P_j\right)B_i, \quad (10)$$

$$E_i = (P_j - P_{j+1})C_i + P_j. \quad (11)$$

5 Fitting Strategy and Algorithm

In this section we will describe the strategy of proposed spline fitting algorithm to video data. Fitting process is applied to temporal data of each pixel individually. Color or luminance value of a pixel at frame i is p_i, where $0 \le p_i \le 255$ and $1 \le i \le n$. We have to approximate the n values of each pixel $O = \{p_1, p_2, \ldots, p_n\}$ (original data) by spline fitting. As an input to algorithm the user specifies two parameters: (1) *upper limit*

Algorithm 1 Fitting Algorithm, pixel level error bound

Require: Points of original data: $O = \{p_1, p_2, \ldots, p_n\}$, Initial set of keyframes $K = \{K_1, K_2, \ldots, K_l\}$, *Maximum allowed square distance* ξ^{lmt}.

1: Fit spline using K, i.e., Find $Q = \{q_1, q_2, \ldots, q_n\}$
2: $d_i^2 = \|p_i - q_i\|^2, \quad 0 \leq i \leq n$
3: $\xi^{max} = Max\left(d_1^2, d_2^2, \ldots, d_n^2\right)$, $\xi^{max} \in j^{th}$ segment
4: **while** $\xi^{max} > \xi^{lmt}$ **do**
5: Split j^{th} segment into j_1^{th} and j_2^{th} segments
6: Add new keyframe $K_{new} = i$ in K
7: Fit spline using updated K, i.e., Find $Q(t_i)$ of $j-1$, j_1, j_2, $j+1$ segments only
8: Update d_i^2 of $j-1$, j_1, j_2, $j+1$ segments only
9: $\xi^{max} = Max\left(d_1^2, d_2^2, \ldots, d_n^2\right)$, $\xi^{max} \in j^{th}$ segment
10: **end while**

of error ξ^{lmt}, i.e., maximum allowed square distance between original and fitted data, e.g., $\xi^{lmt} = 100$ (2) *initial keyframe interval* Δ, i.e., pixel after every Δ^{th} frames is taken as an *end control point* of Cardinal spline, e.g., $\Delta = 12$ then set of initial keyframes K is $K = \{1, 13, 25, 37, \ldots, n\}$ (last frame is always taken as keyframe). The fitting process divides the data into segments based on keyframes. A segment is set of all points (pixels) between two consecutive keyframes, e.g., $S_1 = \{p_1, p_2, \ldots, p_{13}\}$, $S_2 = \{p_{13}, p_{14}, \ldots, p_{25}\}$. Spline interpolation is performed for each segment and n interpolated values (approximated data) $Q = \{q_1, q_2, \ldots, q_n\}$ are obtained (common interpolated values are removed at the joint of two segments). Then we compute the error, i.e., squared distance between each point of original and its corresponding point on approximated data $d_i^2 = \|p_i - q_i\|^2$, $1 \leq i \leq n$. Among all error values we compute the maximum error $\xi^{max} = Max\left(d_1^2, d_2^2, \ldots, d_n^2\right)$. If ξ^{max} is greater than ξ^{lmt} then we add a point (new keyframe) from original data in the set of keyframes where the error is maximum between original and approximated data. Due to addition of a new keyframe a segment is split and replaced by two new segments. For example if $\xi^{max} = d_6^2$ then segment S_1 is split and a new keyframe 6 is inserted between keyframes 1 and 13 ($K = \{1, 6, 13, 25, 37, \ldots, n\}$) and two new segments $\{p_1, p_2, \ldots, p_6\}$ and $\{p_6, p_7, \ldots, p_{13}\}$ replace S_1. The fitting process is repeated with new set of keyframes until ξ^{max} is less than or equal to ξ^{lmt} for each segment. The Algorithm 1 describes the proposed algorithm formally. Figure 3 shows fitting of a pixel data by Cardinal spline.

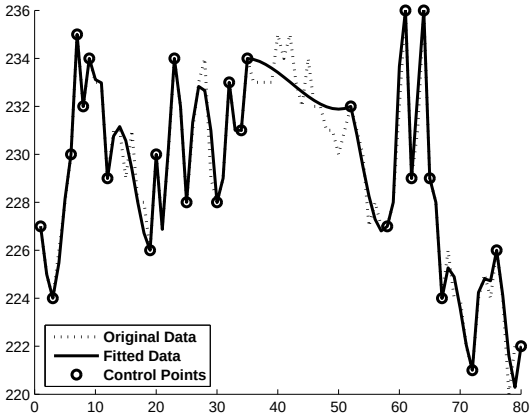

Figure 3: Cardinal spline least square fitting of pixel data in 80 frames of a video.

6 Bitstream Construction and Parsing

This section describes a compact construction of output bitstream and an algorithm to parse the bitstream. The output data of Cardinal spline for each segment consist of (a) *control points* (b) *Tension* (c) *count of interpolating points*. Since *control points* are nothing but pixel values of keyframes from original video data, therefore number of bits allocated to a *control point* are equal to bit per pixel of original video data. Let a *control point* needs b_1 bits and let bit sequences of a *control point* is denoted by B_1 then $b_1 = 8$ (e.g., $B_1 = 10001111$). *Tension* value is usually less than control point value but it can be negative. Let *Tension* needs b_2 bits and let bit sequences of a *Tension* is denoted by B_2. The most significant bit of B_2 is reserved for sign, 0 for positive and 1 for negative. If value of *Tension* value is exceeds b_2 bits, it is truncated to limit (this rarely happens). We allocated $b_2 = 8$ (e.g., $B_1 = 01001111$). Let each count of interpolation points between adjacent keyframes needs b_3 bits. Let bit sequence of count of interpolation points is denoted by B_3. For example count of interpolating points between keyframe 25 and keyframe 37 is 13 (=37-25+1). If the initial interval between keyframes is Δ then $b_3 = \lceil \log_2(\Delta+1) \rceil$, because there must be a keyframe before (in case of sub-segmentation) or after every Δ^{th} keyframe and consequently frame count between adjacent keyframes is $\leq (\Delta+1)$, which can be represented by $\lceil \log_2(\Delta+1) \rceil$ bits. We take keyframe after every 12^{th} frame, therefore we allocated $b_3 = 4$ (e.g. $B_3 = 0110$). It is important to note that even if two consecutive frames e.g. 13 and 14 are keyframes (due to splitting) then count of interpolating points is 2(=14-13+1), it means minimum

value of $B_3 = 0010$ and all bits of B_3 can never be zero.

The fitted data is a continuous curve between adjacent segments. Therefore last control point of i^{th} segment is the the first control point of $(i+1)^{th}$ segment. We need to save only the last control points of all segments except first segment. Let a pixel approximation consists of u segments, $1 \leq s \leq u$.

Now we have sufficient information to define the bitstream syntax of output data. Let there are Z $(= W \times H)$ pixels whose temporal variations are to be approximated. Let u_i is total number of segments obtained from temporal data approximation of i^{th} pixel, $1 \leq i \leq Z$. Let B is the continuous bitstream of output data (for all pixels). Then syntax of B is defined as follows:

$$
\begin{aligned}
B \;=\; & \underbrace{\overbrace{B_1B_1B_2B_3}^{s=1}\overbrace{B_1B_2B_3}^{s=2}\cdots\overbrace{B_1B_2B_3}^{s=u_1}}_{i=1} \\
& \underbrace{\overbrace{B_1B_1B_2B_3}^{s=1}\overbrace{B_1B_2B_3}^{s=2}\cdots\overbrace{B_1B_2B_3}^{s=u_2}}_{i=2} \\
& \cdots \\
& \cdots \\
& \underbrace{\overbrace{B_1B_1B_2B_3}^{s=1}\overbrace{B_1B_2B_3}^{s=2}\cdots\overbrace{B_1B_2B_3}^{s=u_Z}}_{i=Z}.
\end{aligned}
$$

We reserved $B_3 = 0000$ for identification of last segment of current pixel. During parsing of bitstream we check either $B_3 = 0000$ or not. No extra bits are allocated for count of interpolating points of last segment. In order to find the count of interpolating points of last segment of current pixel we have to sum up count of interpolating points before the last segment of each pixel. Then using this sum, we can find the of count of interpolating points for the last segment of current pixel by subtracting the sum from count of frames in video, which is constant for all pixels.

$$
\overbrace{B_3}^{s=u} = n - \left(\overbrace{B_3 - 1}^{s=1} + \overbrace{B_3 - 1}^{s=2} + \cdots + \overbrace{B_3 - 1}^{s=u-1} \right). \quad (12)
$$

In Eq. (12) we subtracted 1 from each B_3 term because not to count the common keyframe between each adjacent segments. Once the next pixel segmentation begins we reset the sum counter for next pixel. This special treatment of last segment of each pixel saves 4-bits per pixel segmentation, which is worth to care. Algorithm 2 is the pseudocode of bitstream parsing:

7 Results

Extensive simulation is done on various standard test video sequences using proposed algorithm. The performance of the method is evaluated in terms of *PSNR*, and *Bitrate* (measured in bits per pixel (*bpp*)) of approximated video data. High *PSNR* and less *Bitrate* are desirable. Table 2 compares the performance of proposed method with well known Three Step Search (TSS) block matching algorithm for luminance videos of Table 1. Our objective is to compare the temporal compression of proposed with temporal compression of existing methods like TSS. Therefor for both proposed methods and TSS output data is coded without spatial compression techniques (e.g., Discrete Cosine Transform (DCT) or Wavelet Transform), without entropy encoding (e.g., arithmetic/Huffman coding) and without quantization. It can be observe from the table that proposed method yields better *PSNR* at lower *Bitrate*. Figure 4 and Fig. 5 show variation of *PSNR* and *Bitrate* at various values of ξ for input sequences. Figure 6 and Fig. 7 show original and approximated frames of input sequences.

Algorithm 2 Bitstream Parsing

```
1:  c ← 0 {count of intrp. points for current pixel}
2:  f1 ← 1 {flag to identify first segment}
3:  f2 ← 0 {flag to identify last segment}
4:  pixel ← 1
5:  while !(End of Bitstream) do
6:      if f1 == 1 then
7:          Read B1B1B2B3
8:          f1 ← 0
9:      else
10:         Read B1B2B3
11:     end if
12:     if B3 == 0000 then
13:         B3 ← n − c
14:         f2 ← 1
15:     else
16:         c ← c + B3 − 1
17:     end if
18:     if f2 == 1 then
19:         pixel ← pixel + 1, c ← 0, f1 ← 1, f2 ← 0
20:     end if
21: end while
```

Table 1: Details of input video sequences used in simulation.

Video Name	Frame Size	Number of Frames
Darius	352×288	25
Football	352×240	25

Table 2: Comparison of proposed method with TSS method. $\Delta = 12$, $b_1 = 8$, $b_2 = 8$, $b_3 = 4$.

	TSS Method		Proposed Method	
Video Name	*Bitrate* (bpp)	*PSNR* (dB)	*Bitrate* (bpp)	*PSNR* (dB)
Darius	4.17	41.5734	4.1227	51.807
Football	4.175	29.7588	4.0776	37.13

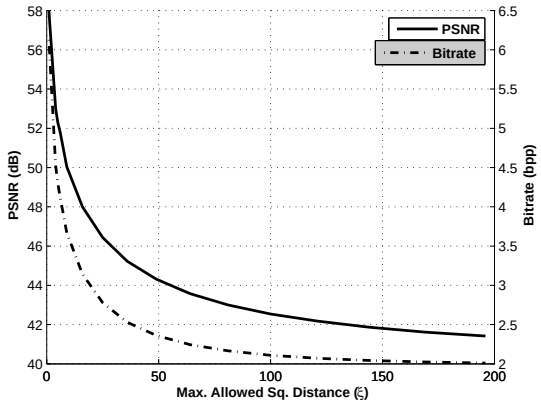

Figure 4: *PSNR* and *Bitrate* performance of *Darius* sequence with varying ξ.

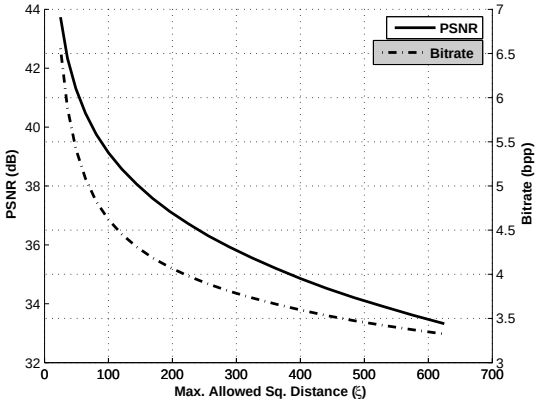

Figure 5: *PSNR* and *Bitrate* performance of *Football* sequence with varying value of ξ.

8 Discussion

***Split of a segment*:** Lines 5 to 8 are important steps of Algorithm 1. For correct and efficient implementation of algorithm it is very important to find which other segments are effected by splitting of a segment. Let assume an arbitrary segment j going to be splits into two segments j_1 and j_2. For three consecutive segments $j-1$, j and $j+1$. The last three control points of $(j-1)^{th}$ segment i.e., P_{j-1}, P_j and P_{j+1} are same as first three control point of j^{th} segment and first control point of $(j-1)^{th}$ segment i.e P_{j-2} is not shared with j^{th} segment. Similarly the first three control points of $(j+1)^{th}$ segment i.e., P_j, P_{j+1} and P_{j+2} are same as last three control points of j^{th} segment and last control point of $(j+1)^{th}$ segment i.e P_{j+3} is not shared with j^{th} segment. This means a segment shares control points with its previous and next segments. Consequently adding a new breakpoint (splitting) at a segment j would require to recompute interpolated values for two new segments j_1 and j_2, obtained from splitting of j^{th} segment, in addition to that previous and next segments of j^{th} segments are also need to be interpolated again.

***Reasons to choose Cardinal spline*:** There are many reasons to choose Cardinal spline for our proposed method: (1) Its implementation is simple and efficient (2) Least square solution requires to save one parameter i.e,. *Tension* in Cardinal spline, whereas in cubic Bézer least square solution requires to save two *middle control points* (3) *Tension* parameter is independent of data dimension, i.e., it is always scalar (1–*D*), whereas for other spline/curves (coefficients/control points) are as of same dimension as of original data. This means for videos in (3–*D*) *RGB* color-space, *Tension* is (1–*D*) but (coefficients/control points) of other splines are in (3–*D*). (4) Generally value of *Tension* is small and can be coded using less number of bits.

***Decoding*:** Decoding is simple interpolation via Eq. (3) using saved *control points*, *count of interpolating points* and *Tension* values.

9 Conclusion

We presented a method for lossy compression of temporal video data by Cardinal spline least square fitting at pixel level. Compact construction of output bitstream and an algorithm to parse the bitstream are also described. The proposed method fully automates the compression/decompression process. The method can be applied to video data of any dimension. Experimental results show that the proposed method

Figure 6: 8^{th} RGB frame of original (top) and approximated (bottom) *Darius* video sequence. $\Delta = 12$, $\xi = 625$.

Figure 7: 8^{th} luminance frame of original (top) and approximated (bottom) *Football* video sequence. $\Delta = 12$, $\xi = 625$.

yields very good results both in terms of objective and subjective quality measurement parameters i.e. *Bitrate*/*PSNR* and *human visual acceptance* without causing any blocking artifacts.

References

[1] A. Bejancu. A new approach to semi-cardinal spline interpolation. *East Journal on Approximations*, 6(4):447–463, 2000.

[2] De Boor C. *A Practical Guide to Splines*. Springer-Verlag, New York, 1978.

[3] Koichi Itoh and Yoshio Ohno. A curve fitting algorithm for character fonts. *Electronic Publishing*, 6(3):195–198, 1993.

[4] Xuan Jing and Lap-Pui Chau. An efficient three-step search algorithm for block motion estimation. *IEEE Transactions on Multimedia*, 6(3):438–442, June 2004.

[5] Doris H.U. Kochanek. Interpolating splines with local tension, continuity, and bias control. *Computer Graphics*, 18(3):33–41, July 1984.

[6] Jianhua Lu and Ming L. Liou. A simple and efficient search algorithm for block-matching motion estimation. *IEEE Transactions on Circuits and Systems for Video Technology*, 7(2):429–433, April 1997.

[7] Editors Martine J. Silbermann, Hemant D. Tagare. Local cardinal spline interpolation and its application to image processing. *Proceedings of SPIE*, pages 272–283, February 1992.

[8] Atsushi Miyazawa, Motonaga Ishii, and Kazunori Okuzawa. Future 3*D* television broadcasting system in a very primitive form. *3D Image Conference*, 2005.

[9] M. Sarfraz and M.A. Khan. An automatic algorithm for approximating boundary of bitmap characters. *Elsevier, Future Generation Computer Systems*, 20(8):1327–1336, 2004.

[10] Y. Q. Shi and X. Xia. A thresholding multiresolution block matching algorithm. *IEEE Transactions on Circuits and Systems for Video Technology*, 7(2):437–440, April 1997.